



PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 3.0

OASIS Standard

15 June 2020

This stage:

<https://docs.oasis-open.org/pkcs11/pkcs11-curr/v3.0/os/pkcs11-curr-v3.0-os.docx> (Authoritative)

<https://docs.oasis-open.org/pkcs11/pkcs11-curr/v3.0/os/pkcs11-curr-v3.0-os.html>

<https://docs.oasis-open.org/pkcs11/pkcs11-curr/v3.0/os/pkcs11-curr-v3.0-os.pdf>

Previous stage:

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

<https://docs.oasis-open.org/pkcs11/pkcs11-curr/v3.0/cs01/pkcs11-curr-v3.0-cs01.html>

<https://docs.oasis-open.org/pkcs11/pkcs11-curr/v3.0/cs01/pkcs11-curr-v3.0-cs01.pdf>

Latest stage:

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

<https://docs.oasis-open.org/pkcs11/pkcs11-curr/v3.0/pkcs11-curr-v3.0.html>

<https://docs.oasis-open.org/pkcs11/pkcs11-curr/v3.0/pkcs11-curr-v3.0.pdf>

Technical Committee:

OASIS PKCS 11 TC

Chairs:

Tony Cox (tony.cox@cryptsoft.com), Cryptsoft Pty Ltd

Robert Relyea (rrelyea@redhat.com), Red Hat

Editors:

Chris Zimman (chris@wmpp.com), Individual

Dieter Bong (dieter.bong@utimaco.com), Utimaco IS GmbH

Additional artifacts:

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

- PKCS #11 header files:
<https://docs.oasis-open.org/pkcs11/pkcs11-curr/v3.0/os/include/pkcs11-v3.0/>
- **ALERT:** Due to a clerical error when publishing the Committee Specification, the header files listed above are outdated and may contain serious flaws. The TC is addressing this in the next round of edits. Meanwhile, users of the standard can find the correct header files at <https://github.com/oasis-tcs/pkcs11/tree/master/working/3-00-current>.

Related work:

This specification replaces or supersedes:

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

This specification is related to:

- *PKCS #11 Cryptographic Token Interface Profiles Version 3.0*. Edited by Tim Hudson. Latest stage. <https://docs.oasis-open.org/pkcs11/pkcs11-profiles/v3.0/pkcs11-profiles-v3.0.html>.

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

Abstract:

This document defines data types, functions and other basic components of the PKCS #11 Cryptoki interface.

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 stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=pkcs11#technical.

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

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

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

Citation format:

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

[PKCS11-Current-v3.0]

PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 3.0. Edited by Chris Zimman and Dieter Bong. 15 June 2020. OASIS Standard. <https://docs.oasis-open.org/pkcs11/pkcs11-curr/v3.0/os/pkcs11-curr-v3.0-os.html>. Latest stage: <https://docs.oasis-open.org/pkcs11/pkcs11-curr/v3.0/pkcs11-curr-v3.0.html>.

Notices

Copyright © OASIS Open 2020. 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	15
1.1	IPR Policy	15
1.2	Terminology	15
1.3	Definitions	15
1.4	Normative References	17
1.5	Non-Normative References	18
2	Mechanisms	21
2.1	RSA.....	21
2.1.1	Definitions.....	22
2.1.2	RSA public key objects.....	23
2.1.3	RSA private key objects	24
2.1.4	PKCS #1 RSA key pair generation	25
2.1.5	X9.31 RSA key pair generation.....	26
2.1.6	PKCS #1 v1.5 RSA	26
2.1.7	PKCS #1 RSA OAEP mechanism parameters	27
2.1.8	PKCS #1 RSA OAEP	28
2.1.9	PKCS #1 RSA PSS mechanism parameters	29
2.1.10	PKCS #1 RSA PSS	29
2.1.11	ISO/IEC 9796 RSA.....	30
2.1.12	X.509 (raw) RSA	31
2.1.13	ANSI X9.31 RSA	32
2.1.14	PKCS #1 v1.5 RSA signature with MD2, MD5, SHA-1, SHA-256, SHA-384, SHA-512, RIPE-MD 128 or RIPE-MD 160	32
2.1.15	PKCS #1 v1.5 RSA signature with SHA-224	33
2.1.16	PKCS #1 RSA PSS signature with SHA-224	33
2.1.17	PKCS #1 RSA PSS signature with SHA-1, SHA-256, SHA-384 or SHA-512.....	33
2.1.18	PKCS #1 v1.5 RSA signature with SHA3.....	34
2.1.19	PKCS #1 RSA PSS signature with SHA3	34
2.1.20	ANSI X9.31 RSA signature with SHA-1	34
2.1.21	TPM 1.1b and TPM 1.2 PKCS #1 v1.5 RSA	34
2.1.22	TPM 1.1b and TPM 1.2 PKCS #1 RSA OAEP	35
2.1.23	RSA AES KEY WRAP	36
2.1.24	RSA AES KEY WRAP mechanism parameters	37
2.1.25	FIPS 186-4	37
2.2	DSA.....	37
2.2.1	Definitions.....	38
2.2.2	DSA public key objects.....	39
2.2.3	DSA Key Restrictions	40
2.2.4	DSA private key objects	40
2.2.5	DSA domain parameter objects	41
2.2.6	DSA key pair generation	42
2.2.7	DSA domain parameter generation.....	42
2.2.8	DSA probabilistic domain parameter generation.....	42
2.2.9	Shawe-Taylor domain parameter generation	43

2.2.10 DSA base domain parameter generation	43
2.2.11 DSA without hashing	43
2.2.12 DSA with SHA-1	44
2.2.13 FIPS 186-4	44
2.2.14 DSA with SHA-224	44
2.2.15 DSA with SHA-256	45
2.2.16 DSA with SHA-384	45
2.2.17 DSA with SHA-512	46
2.2.18 DSA with SHA3-224	46
2.2.19 DSA with SHA3-256	47
2.2.20 DSA with SHA3-384	47
2.2.21 DSA with SHA3-512	47
2.3 Elliptic Curve	48
2.3.1 EC Signatures	50
2.3.2 Definitions	50
2.3.3 ECDSA public key objects	51
2.3.4 Elliptic curve private key objects	52
2.3.5 Edwards Elliptic curve public key objects	54
2.3.6 Edwards Elliptic curve private key objects	54
2.3.7 Montgomery Elliptic curve public key objects	55
2.3.8 Montgomery Elliptic curve private key objects	56
2.3.9 Elliptic curve key pair generation	57
2.3.10 Edwards Elliptic curve key pair generation	58
2.3.11 Montgomery Elliptic curve key pair generation	58
2.3.12 ECDSA without hashing	59
2.3.13 ECDSA with hashing	59
2.3.14 EdDSA	60
2.3.15 XEdDSA	60
2.3.16 EC mechanism parameters	61
2.3.17 Elliptic curve Diffie-Hellman key derivation	66
2.3.18 Elliptic curve Diffie-Hellman with cofactor key derivation	67
2.3.19 Elliptic curve Menezes-Qu-Vanstone key derivation	67
2.3.20 ECDH AES KEY WRAP	68
2.3.21 ECDH AES KEY WRAP mechanism parameters	69
2.3.22 FIPS 186-4	70
2.4 Diffie-Hellman	70
2.4.1 Definitions	71
2.4.2 Diffie-Hellman public key objects	71
2.4.3 X9.42 Diffie-Hellman public key objects	72
2.4.4 Diffie-Hellman private key objects	72
2.4.5 X9.42 Diffie-Hellman private key objects	73
2.4.6 Diffie-Hellman domain parameter objects	74
2.4.7 X9.42 Diffie-Hellman domain parameters objects	75
2.4.8 PKCS #3 Diffie-Hellman key pair generation	76
2.4.9 PKCS #3 Diffie-Hellman domain parameter generation	76
2.4.10 PKCS #3 Diffie-Hellman key derivation	76

2.4.11 X9.42 Diffie-Hellman mechanism parameters.....	77
2.4.12 X9.42 Diffie-Hellman key pair generation.....	80
2.4.13 X9.42 Diffie-Hellman domain parameter generation	81
2.4.14 X9.42 Diffie-Hellman key derivation	81
2.4.15 X9.42 Diffie-Hellman hybrid key derivation	81
2.4.16 X9.42 Diffie-Hellman Menezes-Qu-Vanstone key derivation.....	82
2.5 Extended Triple Diffie-Hellman (x3dh).....	83
2.5.1 Definitions.....	83
2.5.2 Extended Triple Diffie-Hellman key objects	83
2.5.3 Initiating an Extended Triple Diffie-Hellman key exchange.....	83
2.5.4 Responding to an Extended Triple Diffie-Hellman key exchange.....	84
2.5.5 Extended Triple Diffie-Hellman parameters	85
2.6 Double Ratchet	85
2.6.1 Definitions.....	86
2.6.2 Double Ratchet secret key objects.....	86
2.6.3 Double Ratchet key derivation	87
2.6.4 Double Ratchet Encryption mechanism	88
2.6.5 Double Ratchet parameters	88
2.7 Wrapping/unwrapping private keys	89
2.8 Generic secret key	91
2.8.1 Definitions.....	91
2.8.2 Generic secret key objects	92
2.8.3 Generic secret key generation	92
2.9 HMAC mechanisms	93
2.9.1 General block cipher mechanism parameters.....	93
2.10 AES.....	93
2.10.1 Definitions.....	93
2.10.2 AES secret key objects	94
2.10.3 AES key generation.....	95
2.10.4 AES-ECB.....	95
2.10.5 AES-CBC.....	95
2.10.6 AES-CBC with PKCS padding	96
2.10.7 AES-OFB.....	97
2.10.8 AES-CFB	97
2.10.9 General-length AES-MAC	98
2.10.10 AES-MAC	98
2.10.11 AES-XCBC-MAC	98
2.10.12 AES-XCBC-MAC-96.....	98
2.11 AES with Counter	99
2.11.1 Definitions.....	99
2.11.2 AES with Counter mechanism parameters	99
2.11.3 AES with Counter Encryption / Decryption.....	100
2.12 AES CBC with Cipher Text Stealing CTS.....	100
2.12.1 Definitions.....	100
2.12.2 AES CTS mechanism parameters	100
2.13 Additional AES Mechanisms	101

2.13.1 Definitions.....	101
2.13.2 AES-GCM Authenticated Encryption / Decryption	101
2.13.3 AES-CCM authenticated Encryption / Decryption.....	103
2.13.4 AES-GMAC	105
2.13.5 AES GCM and CCM Mechanism parameters.....	105
2.14 AES CMAC	108
2.14.1 Definitions.....	108
2.14.2 Mechanism parameters	108
2.14.3 General-length AES-CMAC.....	108
2.14.4 AES-CMAC.....	109
2.15 AES XTS.....	109
2.15.1 Definitions.....	109
2.15.2 AES-XTS secret key objects	110
2.15.3 AES-XTS key generation	110
2.15.4 AES-XTS	110
2.16 AES Key Wrap	110
2.16.1 Definitions.....	111
2.16.2 AES Key Wrap Mechanism parameters.....	111
2.16.3 AES Key Wrap	111
2.17 Key derivation by data encryption – DES & AES	111
2.17.1 Definitions.....	112
2.17.2 Mechanism Parameters	112
2.17.3 Mechanism Description	112
2.18 Double and Triple-length DES	113
2.18.1 Definitions.....	113
2.18.2 DES2 secret key objects	113
2.18.3 DES3 secret key objects	114
2.18.4 Double-length DES key generation	115
2.18.5 Triple-length DES Order of Operations	115
2.18.6 Triple-length DES in CBC Mode	115
2.18.7 DES and Triple length DES in OFB Mode	115
2.18.8 DES and Triple length DES in CFB Mode.....	116
2.19 Double and Triple-length DES CMAC	116
2.19.1 Definitions.....	117
2.19.2 Mechanism parameters.....	117
2.19.3 General-length DES3-MAC	117
2.19.4 DES3-CMAC	117
2.20 SHA-1	118
2.20.1 Definitions.....	118
2.20.2 SHA-1 digest	118
2.20.3 General-length SHA-1-HMAC	119
2.20.4 SHA-1-HMAC	119
2.20.5 SHA-1 key derivation.....	119
2.20.6 SHA-1 HMAC key generation.....	120
2.21 SHA-224	120
2.21.1 Definitions.....	120

2.21.2 SHA-224 digest	121
2.21.3 General-length SHA-224-HMAC	121
2.21.4 SHA-224-HMAC	121
2.21.5 SHA-224 key derivation.....	121
2.21.6 SHA-224 HMAC key generation.....	121
2.22 SHA-256	122
2.22.1 Definitions.....	122
2.22.2 SHA-256 digest	122
2.22.3 General-length SHA-256-HMAC	122
2.22.4 SHA-256-HMAC	123
2.22.5 SHA-256 key derivation.....	123
2.22.6 SHA-256 HMAC key generation.....	123
2.23 SHA-384	123
2.23.1 Definitions.....	124
2.23.2 SHA-384 digest	124
2.23.3 General-length SHA-384-HMAC	124
2.23.4 SHA-384-HMAC	125
2.23.5 SHA-384 key derivation.....	125
2.23.6 SHA-384 HMAC key generation.....	125
2.24 SHA-512	125
2.24.1 Definitions.....	126
2.24.2 SHA-512 digest	126
2.24.3 General-length SHA-512-HMAC	126
2.24.4 SHA-512-HMAC	126
2.24.5 SHA-512 key derivation.....	127
2.24.6 SHA-512 HMAC key generation.....	127
2.25 SHA-512/224	127
2.25.1 Definitions.....	127
2.25.2 SHA-512/224 digest	127
2.25.3 General-length SHA-512/224-HMAC	128
2.25.4 SHA-512/224-HMAC	128
2.25.5 SHA-512/224 key derivation.....	128
2.25.6 SHA-512/224 HMAC key generation	128
2.26 SHA-512/256	129
2.26.1 Definitions.....	129
2.26.2 SHA-512/256 digest	129
2.26.3 General-length SHA-512/256-HMAC	130
2.26.4 SHA-512/256-HMAC	130
2.26.5 SHA-512/256 key derivation.....	130
2.26.6 SHA-512/256 HMAC key generation	130
2.27 SHA-512/t	131
2.27.1 Definitions.....	131
2.27.2 SHA-512/t digest	131
2.27.3 General-length SHA-512/t-HMAC	131
2.27.4 SHA-512/t-HMAC	132
2.27.5 SHA-512/t key derivation.....	132

2.27.6 SHA-512/t HMAC key generation	132
2.28 SHA3-224	132
2.28.1 Definitions	132
2.28.2 SHA3-224 digest	133
2.28.3 General-length SHA3-224-HMAC	133
2.28.4 SHA3-224-HMAC	133
2.28.5 SHA3-224 key derivation	133
2.28.6 SHA3-224 HMAC key generation	133
2.29 SHA3-256	134
2.29.1 Definitions	134
2.29.2 SHA3-256 digest	134
2.29.3 General-length SHA3-256-HMAC	135
2.29.4 SHA3-256-HMAC	135
2.29.5 SHA3-256 key derivation	135
2.29.6 SHA3-256 HMAC key generation	135
2.30 SHA3-384	136
2.30.1 Definitions	136
2.30.2 SHA3-384 digest	136
2.30.3 General-length SHA3-384-HMAC	136
2.30.4 SHA3-384-HMAC	137
2.30.5 SHA3-384 key derivation	137
2.30.6 SHA3-384 HMAC key generation	137
2.31 SHA3-512	137
2.31.1 Definitions	138
2.31.2 SHA3-512 digest	138
2.31.3 General-length SHA3-512-HMAC	138
2.31.4 SHA3-512-HMAC	138
2.31.5 SHA3-512 key derivation	139
2.31.6 SHA3-512 HMAC key generation	139
2.32 SHAKE	139
2.32.1 Definitions	139
2.32.2 SHAKE Key Derivation	139
2.33 Blake2b-160	140
2.33.1 Definitions	140
2.33.2 BLAKE2B-160 digest	140
2.33.3 General-length BLAKE2B-160-HMAC	141
2.33.4 BLAKE2B-160-HMAC	141
2.33.5 BLAKE2B-160 key derivation	141
2.33.6 BLAKE2B-160 HMAC key generation	141
2.34 BLAKE2B-256	141
2.34.1 Definitions	142
2.34.2 BLAKE2B-256 digest	142
2.34.3 General-length BLAKE2B-256-HMAC	142
2.34.4 BLAKE2B-256-HMAC	143
2.34.5 BLAKE2B-256 key derivation	143
2.34.6 BLAKE2B-256 HMAC key generation	143

2.35 BLAKE2B-384.....	143
2.35.1 Definitions	144
2.35.2 BLAKE2B-384 digest.....	144
2.35.3 General-length BLAKE2B-384-HMAC	144
2.35.4 BLAKE2B-384-HMAC	144
2.35.5 BLAKE2B-384 key derivation	145
2.35.6 BLAKE2B-384 HMAC key generation.....	145
2.36 BLAKE2B-512.....	145
2.36.1 Definitions	145
2.36.2 BLAKE2B-512 digest.....	145
2.36.3 General-length BLAKE2B-512-HMAC	146
2.36.4 BLAKE2B-512-HMAC	146
2.36.5 BLAKE2B-512 key derivation	146
2.36.6 BLAKE2B-512 HMAC key generation.....	146
2.37 PKCS #5 and PKCS #5-style password-based encryption (PBE).....	147
2.37.1 Definitions	147
2.37.2 Password-based encryption/authentication mechanism parameters.....	147
2.37.3 PKCS #5 PBKDF2 key generation mechanism parameters	148
2.37.4 PKCS #5 PBKD2 key generation	150
2.38 PKCS #12 password-based encryption/authentication mechanisms	150
2.38.1 SHA-1-PBE for 3-key triple-DES-CBC	151
2.38.2 SHA-1-PBE for 2-key triple-DES-CBC	151
2.38.3 SHA-1-PBA for SHA-1-HMAC.....	151
2.39 SSL	152
2.39.1 Definitions	152
2.39.2 SSL mechanism parameters	152
2.39.3 Pre-master key generation	154
2.39.4 Master key derivation	155
2.39.5 Master key derivation for Diffie-Hellman	155
2.39.6 Key and MAC derivation.....	156
2.39.7 MD5 MACing in SSL 3.0	157
2.39.8 SHA-1 MACing in SSL 3.0	157
2.40 TLS 1.2 Mechanisms	158
2.40.1 Definitions	158
2.40.2 TLS 1.2 mechanism parameters	158
2.40.3 TLS MAC	161
2.40.4 Master key derivation	162
2.40.5 Master key derivation for Diffie-Hellman	162
2.40.6 Key and MAC derivation.....	163
2.40.7 CKM_TLS12_KEY_SAFE_DERIVE.....	164
2.40.8 Generic Key Derivation using the TLS PRF.....	164
2.40.9 Generic Key Derivation using the TLS12 PRF	165
2.41 WTLS	166
2.41.1 Definitions	166
2.41.2 WTLS mechanism parameters.....	166
2.41.3 Pre master secret key generation for RSA key exchange suite.....	169

2.41.4 Master secret key derivation	170
2.41.5 Master secret key derivation for Diffie-Hellman and Elliptic Curve Cryptography	170
2.41.6 WTLS PRF (pseudorandom function)	171
2.41.7 Server Key and MAC derivation	171
2.41.8 Client key and MAC derivation	172
2.42 SP 800-108 Key Derivation	173
2.42.1 Definitions	173
2.42.2 Mechanism Parameters	174
2.42.3 Counter Mode KDF	179
2.42.4 Feedback Mode KDF	180
2.42.5 Double Pipeline Mode KDF	180
2.42.6 Deriving Additional Keys	181
2.42.7 Key Derivation Attribute Rules	182
2.42.8 Constructing PRF Input Data	182
2.42.8.1 Sample Counter Mode KDF.....	183
2.42.8.2 Sample SCP03 Counter Mode KDF.....	184
2.42.8.3 Sample Feedback Mode KDF.....	185
2.42.8.4 Sample Double-Pipeline Mode KDF	186
2.43 Miscellaneous simple key derivation mechanisms	187
2.43.1 Definitions	187
2.43.2 Parameters for miscellaneous simple key derivation mechanisms.....	187
2.43.3 Concatenation of a base key and another key.....	188
2.43.4 Concatenation of a base key and data.....	189
2.43.5 Concatenation of data and a base key.....	189
2.43.6 XORing of a key and data	190
2.43.7 Extraction of one key from another key.....	191
2.44 CMS	191
2.44.1 Definitions	192
2.44.2 CMS Signature Mechanism Objects	192
2.44.3 CMS mechanism parameters.....	192
2.44.4 CMS signatures	193
2.45 Blowfish.....	194
2.45.1 Definitions	195
2.45.2 BLOWFISH secret key objects.....	195
2.45.3 Blowfish key generation	196
2.45.4 Blowfish-CBC	196
2.45.5 Blowfish-CBC with PKCS padding	196
2.46 Twofish.....	197
2.46.1 Definitions	197
2.46.2 Twofish secret key objects	197
2.46.3 Twofish key generation	198
2.46.4 Twofish -CBC	198
2.46.5 Twofish-CBC with PKCS padding	198
2.47 CAMELLIA	198
2.47.1 Definitions	199
2.47.2 Camellia secret key objects.....	199

2.47.3 Camellia key generation.....	200
2.47.4 Camellia-ECB	200
2.47.5 Camellia-CBC.....	201
2.47.6 Camellia-CBC with PKCS padding	201
2.47.7 CAMELLIA with Counter mechanism parameters.....	202
2.47.8 General-length Camellia-MAC	203
2.47.9 Camellia-MAC	203
2.48 Key derivation by data encryption - Camellia	203
2.48.1 Definitions.....	203
2.48.2 Mechanism Parameters	204
2.49 ARIA.....	204
2.49.1 Definitions.....	204
2.49.2 Aria secret key objects	205
2.49.3 ARIA key generation	205
2.49.4 ARIA-ECB.....	205
2.49.5 ARIA-CBC	206
2.49.6 ARIA-CBC with PKCS padding	207
2.49.7 General-length ARIA-MAC	207
2.49.8 ARIA-MAC	208
2.50 Key derivation by data encryption - ARIA.....	208
2.50.1 Definitions.....	208
2.50.2 Mechanism Parameters	208
2.51 SEED	209
2.51.1 Definitions.....	210
2.51.2 SEED secret key objects.....	210
2.51.3 SEED key generation	211
2.51.4 SEED-ECB	211
2.51.5 SEED-CBC	211
2.51.6 SEED-CBC with PKCS padding.....	211
2.51.7 General-length SEED-MAC.....	211
2.51.8 SEED-MAC.....	211
2.52 Key derivation by data encryption - SEED	212
2.52.1 Definitions.....	212
2.52.2 Mechanism Parameters	212
2.53 OTP.....	212
2.53.1 Usage overview.....	212
2.53.2 Case 1: Generation of OTP values	213
2.53.3 Case 2: Verification of provided OTP values	214
2.53.4 Case 3: Generation of OTP keys	214
2.53.5 OTP objects.....	215
2.53.5.1 Key objects	215
2.53.6 OTP-related notifications	218
2.53.7 OTP mechanisms	218
2.53.7.1 OTP mechanism parameters	218
2.53.8 RSA SecurID	222
2.53.8.1 RSA SecurID secret key objects	222

2.53.8.2 RSA SecurID key generation	223
2.53.8.3 SecurID OTP generation and validation.....	224
2.53.8.4 Return values.....	224
2.53.9 OATH HOTP.....	224
2.53.9.1 OATH HOTP secret key objects	224
2.53.9.2 HOTP key generation	225
2.53.9.3 HOTP OTP generation and validation.....	225
2.53.10 ActivIdentity ACTI.....	225
2.53.10.1 ACTI secret key objects	225
2.53.10.2 ACTI key generation	226
2.53.10.3 ACTI OTP generation and validation.....	226
2.54 CT-KIP	227
2.54.1 Principles of Operation	227
2.54.2 Mechanisms	227
2.54.3 Definitions.....	228
2.54.4 CT-KIP Mechanism parameters	228
2.54.5 CT-KIP key derivation	228
2.54.6 CT-KIP key wrap and key unwrap.....	229
2.54.7 CT-KIP signature generation.....	229
2.55 GOST 28147-89	229
2.55.1 Definitions.....	230
2.55.2 GOST 28147-89 secret key objects	230
2.55.3 GOST 28147-89 domain parameter objects	231
2.55.4 GOST 28147-89 key generation	231
2.55.5 GOST 28147-89-ECB	232
2.55.6 GOST 28147-89 encryption mode except ECB	232
2.55.7 GOST 28147-89-MAC	233
2.55.8 GOST 28147-89 keys wrapping/unwrapping with GOST 28147-89	233
2.56 GOST R 34.11-94.....	234
2.56.1 Definitions.....	234
2.56.2 GOST R 34.11-94 domain parameter objects.....	234
2.56.3 GOST R 34.11-94 digest.....	235
2.56.4 GOST R 34.11-94 HMAC	236
2.57 GOST R 34.10-2001.....	236
2.57.1 Definitions.....	237
2.57.2 GOST R 34.10-2001 public key objects	237
2.57.3 GOST R 34.10-2001 private key objects	238
2.57.4 GOST R 34.10-2001 domain parameter objects.....	240
2.57.5 GOST R 34.10-2001 mechanism parameters.....	241
2.57.6 GOST R 34.10-2001 key pair generation.....	242
2.57.7 GOST R 34.10-2001 without hashing	242
2.57.8 GOST R 34.10-2001 with GOST R 34.11-94	243
2.57.9 GOST 28147-89 keys wrapping/unwrapping with GOST R 34.10-2001	243
2.57.10 Common key derivation with assistance of GOST R 34.10-2001 keys	244
2.58 ChaCha20.....	244
2.58.1 Definitions.....	244
2.58.2 ChaCha20 secret key objects	244

2.58.3 ChaCha20 mechanism parameters	245
2.58.4 ChaCha20 key generation.....	245
2.58.5 ChaCha20 mechanism.....	246
2.59 Salsa20	247
2.59.1 Definitions.....	247
2.59.2 Salsa20 secret key objects.....	247
2.59.3 Salsa20 mechanism parameters.....	248
2.59.4 Salsa20 key generation	248
2.59.5 Salsa20 mechanism	248
2.60 Poly1305	249
2.60.1 Definitions.....	249
2.60.2 Poly1305 secret key objects.....	250
2.60.3 Poly1305 mechanism	250
2.61 Chacha20/Poly1305 and Salsa20/Poly1305 Authenticated Encryption / Decryption.....	250
2.61.1 Definitions.....	251
2.61.2 Usage	251
2.61.3 ChaCha20/Poly1305 and Salsa20/Poly1305 Mechanism parameters	252
2.62 HKDF Mechanisms	253
2.62.1 Definitions.....	254
2.62.2 HKDF mechanism parameters	254
2.62.3 HKDF derive	255
2.62.4 HKDF Data	256
2.62.5 HKDF Key gen	256
2.63 NULL Mechanism	256
2.63.1 Definitions.....	256
2.63.2 CKM_NULL mechanism parameters	256
3 PKCS #11 Implementation Conformance	257
Appendix A. Acknowledgments	258
Appendix B. Manifest Constants	260
Appendix C. Revision History	261

1 Introduction

2 This document defines mechanisms that are anticipated to be used with the current version of PKCS #11.
3 All text is normative unless otherwise labeled.

4 1.1 IPR Policy

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

10 1.2 Terminology

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

14 1.3 Definitions

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

17	AES	<i>Advanced Encryption Standard, as defined in FIPS PUB 197.</i>
18	CAMELLIA	<i>The Camellia encryption algorithm, as defined in RFC 3713.</i>
19	BLOWFISH	<i>The Blowfish Encryption Algorithm of Bruce Schneier, www.schneier.com.</i>
21	CBC	<i>Cipher-Block Chaining mode, as defined in FIPS PUB 81.</i>
22	CDMF	<i>Commercial Data Masking Facility, a block encipherment method specified by International Business Machines Corporation and based on DES.</i>
25	CMAC	<i>Cipher-based Message Authenticate Code as defined in [NIST sp800-38b] and [RFC 4493].</i>
27	CMS	<i>Cryptographic Message Syntax (see RFC 2630)</i>
28	CT-KIP	<i>Cryptographic Token Key Initialization Protocol (as defined in [CT- KIP])</i>
30	DES	<i>Data Encryption Standard, as defined in FIPS PUB 46-3.</i>
31	DSA	<i>Digital Signature Algorithm, as defined in FIPS PUB 186-2.</i>
32	EC	<i>Elliptic Curve</i>
33	ECB	<i>Electronic Codebook mode, as defined in FIPS PUB 81.</i>
34	ECDH	<i>Elliptic Curve Diffie-Hellman.</i>

35	ECDSA	<i>Elliptic Curve DSA, as in ANSI X9.62.</i>
36	ECMQV	<i>Elliptic Curve Menezes-Qu-Vanstone</i>
37	GOST 28147-89	The encryption algorithm, as defined in Part 2 [GOST 28147-89] and [RFC 4357] [RFC 4490], and RFC [4491].
38		
39	GOST R 34.11-94	<i>Hash algorithm, as defined in [GOST R 34.11-94] and [RFC 4357], [RFC 4490], and [RFC 4491].</i>
40		
41	GOST R 34.10-2001	<i>The digital signature algorithm, as defined in [GOST R 34.10-2001] and [RFC 4357], [RFC 4490], and [RFC 4491].</i>
42		
43	IV	<i>Initialization Vector.</i>
44	MAC	<i>Message Authentication Code.</i>
45	MQV	<i>Menezes-Qu-Vanstone</i>
46	OAEP	<i>Optimal Asymmetric Encryption Padding for RSA.</i>
47	PKCS	<i>Public-Key Cryptography Standards.</i>
48	PRF	<i>Pseudo random function.</i>
49	PTD	<i>Personal Trusted Device, as defined in MeT-PTD</i>
50	RSA	<i>The RSA public-key cryptosystem.</i>
51	SHA-1	<i>The (revised) Secure Hash Algorithm with a 160-bit message digest, as defined in FIPS PUB 180-2.</i>
52		
53	SHA-224	<i>The Secure Hash Algorithm with a 224-bit message digest, as defined in RFC 3874. Also defined in FIPS PUB 180-2 with Change Notice 1.</i>
54		
55		
56	SHA-256	<i>The Secure Hash Algorithm with a 256-bit message digest, as defined in FIPS PUB 180-2.</i>
57		
58	SHA-384	<i>The Secure Hash Algorithm with a 384-bit message digest, as defined in FIPS PUB 180-2.</i>
59		
60	SHA-512	<i>The Secure Hash Algorithm with a 512-bit message digest, as defined in FIPS PUB 180-2.</i>
61		
62	SSL	<i>The Secure Sockets Layer 3.0 protocol.</i>
63	SO	<i>A Security Officer user.</i>
64	TLS	<i>Transport Layer Security.</i>
65	WIM	<i>Wireless Identification Module.</i>
66	WTLS	<i>Wireless Transport Layer Security.</i>
67		

68 **1.4 Normative References**

- 69 [ARIA] National Security Research Institute, Korea, "Block Cipher Algorithm ARIA",
70 URL: <http://tools.ietf.org/html/rfc5794>
- 71 [BLOWFISH] B. Schneier. Description of a New Variable-Length Key, 64-Bit Block Cipher
72 (Blowfish), December 1993.
73 URL: <https://www.schneier.com/paper-blowfish-fse.html>
- 74 [CAMELLIA] M. Matsui, J. Nakajima, S. Moriai. A Description of the Camellia Encryption
75 Algorithm, April 2004.
76 URL: <http://www.ietf.org/rfc/rfc3713.txt>
- 77 [CDMF] Johnson, D.B. The Commercial Data Masking Facility (CDMF) data privacy
78 algorithm, March 1994.
79 URL: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5389557>
- 80 [CHACHA] D. Bernstein, ChaCha, a variant of Salsa20, Jan 2008.
81 URL: <http://cr.yp.to/chacha/chacha-20080128.pdf>
- 82 [DH] W. Diffie, M. Hellman. New Directions in Cryptography. Nov, 1976.
83 URL: <http://www-ee.stanford.edu/~hellman/publications/24.pdf>
- 84 [FIPS PUB 81] NIST. FIPS 81: DES Modes of Operation. December 1980.
85 URL: <http://csrc.nist.gov/publications/fips/fips81/fips81.htm>
- 86 [FIPS PUB 186-4] NIST. FIPS 186-4: Digital Signature Standard. July 2013.
87 URL: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>
- 88 [FIPS PUB 197] NIST. FIPS 197: Advanced Encryption Standard. November 26, 2001.
89 URL: <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- 90 [FIPS SP 800-56A] NIST. Special Publication 800-56A Revision 2: *Recommendation for Pair-Wise
91 Key Establishment Schemes Using Discrete Logarithm Cryptography*, May 2013.
92 URL: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf>
- 93 [FIPS SP 800-108] NIST. Special Publication 800-108 (Revised): *Recommendation for Key
94 Derivation Using Pseudorandom Functions*, October 2009.
95 URL: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-108.pdf>
- 96 [GOST] V. Dolmatov, A. Degtyarev. GOST R. 34.11-2012: Hash Function. August 2013.
97 URL: <http://tools.ietf.org/html/rfc6986>
- 98 [MD2] B. Kaliski. RSA Laboratories. The MD2 Message-Digest Algorithm. April, 1992.
99 URL: <http://tools.ietf.org/html/rfc1319>
- 100 [MD5] RSA Data Security. R. Rivest. The MD5 Message-Digest Algorithm. April, 1992.
101 URL: <http://tools.ietf.org/html/rfc1319>
- 102 [OAEP] M. Bellare, P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with
103 RSA. Nov 19, 1995.
104 URL: <http://cseweb.ucsd.edu/users/mihir/papers/oaep.pdf>
- 105 [PKCS11-Base] PKCS #11 Cryptographic Token Interface Base Specification Version 3.0. Edited
106 by Chris Zimman and Dieter Bong. Latest version. <https://docs.oasis-open.org/pkcs11/pkcs11-base/v3.0/pkcs11-base-v3.0.html>.
- 107 [PKCS11-Hist] PKCS #11 Cryptographic Token Interface Historical Mechanisms Specification
108 Version 3.0. Edited by Chris Zimman and Dieter Bong. Latest version.
<https://docs.oasis-open.org/pkcs11/pkcs11-hist/v3.0/pkcs11-hist-v3.0.html>.
- 109 [PKCS11-Prof] PKCS #11 Cryptographic Token Interface Profiles Version 3.0. Edited by Tim
110 Hudson. Latest version. <https://docs.oasis-open.org/pkcs11/pkcs11-profiles/v3.0/pkcs11-profiles-v3.0.html>.
- 111 [POLY1305] D.J. Bernstein. The Poly1305-AES message-authentication code. Jan 2005.
112 URL: <https://cr.yp.to/mac/poly1305-20050329.pdf>
- 113 [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP
114 14, RFC 2119, March 1997.
115 URL: <http://www.ietf.org/rfc/rfc2119.txt>.

120	[RIPEMD]	H. Dobbertin, A. Bosselaers, B. Preneel. The hash function RIPEMD-160, Feb 13, 2012. URL: http://homes.esat.kuleuven.be/~bosselaer/ripemd160.html
121		
122		
123	[SALSA]	D. Bernstein, ChaCha, a variant of Salsa20, Jan 2008. URL: http://cr.yp.to/chacha/chacha-20080128.pdf
124		
125	[SEED]	KISA. SEED 128 Algorithm Specification. Sep 2003. URL: http://seed.kisa.or.kr/html/egovframework/iwt/ds/ko/ref/%5B2%5D_SEED+128_Specification_english_M.pdf
126		
127		
128	[SHA-1]	NIST. FIPS 180-4: Secure Hash Standard. March 2012. URL: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf
129		
130	[SHA-2]	NIST. FIPS 180-4: Secure Hash Standard. March 2012. URL: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf
131		
132	[TWOFISH]	B. Schneier, J. Kelsey, D. Whiting, C. Hall, N. Ferguson. Twofish: A 128-Bit Block Cipher. June 15, 1998. URL: https://www.schneier.com/paper-twofish-paper.pdf
133		
134		

1.5 Non-Normative References

135	[CAP-1.2]	<i>Common Alerting Protocol Version 1.2.</i> 01 July 2010. OASIS Standard. URL: http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html
136		
137		
138	[AES KEYWRAP]	National Institute of Standards and Technology, NIST Special Publication 800-38F, Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping, December 2012, http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf
139		
140		
141		
142	[ANSI C]	ANSI/ISO. American National Standard for Programming Languages – C. 1990.
143	[ANSI X9.31]	Accredited Standards Committee X9. Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA). 1998.
144		
145	[ANSI X9.42]	Accredited Standards Committee X9. Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography. 2003.
146		
147		
148	[ANSI X9.62]	Accredited Standards Committee X9. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). 1998.
149		
150	[ANSI X9.63]	Accredited Standards Committee X9. Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography. 2001. URL: http://webstore.ansi.org/RecordDetail.aspx?sku=X9.63-2011
151		
152		
153		
154	[BRAINPOOL]	ECC Brainpool Standard Curves and Curve Generation, v1.0, 19.10.2005 URL: http://www.ecc-brainpool.org
155		
156	[CT-KIP]	RSA Laboratories. Cryptographic Token Key Initialization Protocol. Version 1.0, December 2005. URL: ftp://ftp.rsasecurity.com/pub/otps/ct-kip/ct-kip-v1-0.pdf .
157		
158		
159	[CC/PP]	CCPP-STRUCT-VOCAB, G. Klyne, F. Reynolds, C. , H. Ohto, J. Hjelm, M. H. Butler, L. Tran, Editors, W3C Recommendation, 15 January 2004, URL: http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/ Latest version available at http://www.w3.org/TR/CCPP-struct-vocab/
160		
161		
162		
163	[LEGIFRANCE]	Avis relatif aux paramètres de courbes elliptiques définis par l'Etat français (Publication of elliptic curve parameters by the French state) URL: https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000024668816
164		
165		
166		
167		
168	[NIST AES CTS]	National Institute of Standards and Technology, Addendum to NIST Special Publication 800-38A, “Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode”
169		
170		

- 171 URL: http://csrc.nist.gov/publications/nistpubs/800-38a/addendum-to-nist_sp800-38A.pdf
- 172
- 173 [PKCS11-UG] PKCS #11 *Cryptographic Token Interface Usage Guide Version 2.41*. Edited by John Leiseboer and Robert Griffin. version: <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html>.
- 174
- 175
- 176 [RFC 2865] Rigney et al, "Remote Authentication Dial In User Service (RADIUS)", IETF RFC2865, June 2000.
URL: <http://www.ietf.org/rfc/rfc2865.txt>.
- 177
- 178
- 179 [RFC 3686] Housley, "Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)," IETF RFC 3686, January 2004.
URL: <http://www.ietf.org/rfc/rfc3686.txt>.
- 180
- 181
- 182 [RFC 3717] Matsui, et al, "A Description of the Camellia Encryption Algorithm," IETF RFC 3717, April 2004.
URL: <http://www.ietf.org/rfc/rfc3713.txt>.
- 183
- 184
- 185 [RFC 3610] Whiting, D., Housley, R., and N. Ferguson, "Counter with CBC-MAC (CCM)", IETF RFC 3610, September 2003.
URL: <http://www.ietf.org/rfc/rfc3610.txt>
- 186
- 187
- 188 [RFC 3874] Smit et al, "A 224-bit One-way Hash Function: SHA-224," IETF RFC 3874, June 2004.
URL: <http://www.ietf.org/rfc/rfc3874.txt>.
- 189
- 190
- 191 [RFC 3748] Aboba et al, "Extensible Authentication Protocol (EAP)", IETF RFC 3748, June 2004.
URL: <http://www.ietf.org/rfc/rfc3748.txt>.
- 192
- 193
- 194 [RFC 4269] South Korean Information Security Agency (KISA) "The SEED Encryption Algorithm", December 2005.
URL: <ftp://ftp.rfc-editor.org/in-notes/rfc4269.txt>
- 195
- 196
- 197 [RFC 4309] Housley, R., "Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)," IETF RFC 4309, December 2005.
URL: <http://www.ietf.org/rfc/rfc4309.txt>
- 198
- 199
- 200 [RFC 4357] V. Popov, I. Kurepkin, S. Leontiev "Additional Cryptographic Algorithms for Use with GOST 28147-89, GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms", January 2006.
URL: <http://www.ietf.org/rfc/rfc4357.txt>
- 201
- 202
- 203
- 204 [RFC 4490] S. Leontiev, Ed. G. Chudov, Ed. "Using the GOST 28147-89, GOST R 34.11-94, GOST R 34.10-94, and GOST R 34.10-2001 Algorithms with Cryptographic Message Syntax (CMS)", May 2006.
URL: <http://www.ietf.org/rfc/rfc4490.txt>
- 205
- 206
- 207
- 208 [RFC 4491] S. Leontiev, Ed., D. Shefanovski, Ed., "Using the GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms with the Internet X.509 Public Key Infrastructure Certificate and CRL Profile", May 2006.
URL: <http://www.ietf.org/rfc/rfc4491.txt>
- 209
- 210
- 211
- 212 [RFC 4493] J. Song et al. *RFC 4493: The AES-CMAC Algorithm*. June 2006.
URL: <http://www.ietf.org/rfc/rfc4493.txt>
- 213
- 214 [RFC 5705] Rescorla, E., "The Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, March 2010.
URL: <http://www.ietf.org/rfc/rfc5705.txt>
- 215
- 216
- 217 [RFC 5869] H. Krawczyk, P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", May 2010
URL: <http://www.ietf.org/rfc/rfc5869.txt>
- 218
- 219
- 220 [RFC 7539] Y Nir, A. Langley. *RFC 7539: ChaCha20 and Poly1305 for IETF Protocols*, May 2015
URL: <https://tools.ietf.org/rfc/rfc7539.txt>
- 221
- 222

223	[RFC 7748]	Aboba et al, "Elliptic Curves for Security", IETF RFC 7748, January 2016 URL: https://tools.ietf.org/html/rfc7748
224	[RFC 8032]	Aboba et al, "Edwards-Curve Digital Signature Algorithm (EdDSA)", IETF RFC 8032, January 2017 URL: https://tools.ietf.org/html/rfc8032
225	[SEC 1]	Standards for Efficient Cryptography Group (SECG). <i>Standards for Efficient Cryptography (SEC) 1: Elliptic Curve Cryptography</i> . Version 1.0, September 20, 2000.
226	[SEC 2]	Standards for Efficient Cryptography Group (SECG). <i>Standards for Efficient Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters</i> . Version 1.0, September 20, 2000.
227	[SIGNAL]	The X3DH Key Agreement Protocol, Revision 1, 2016-11-04, Moxie Marlinspike, Trevor Perrin (editor) URL: https://signal.org/docs/specifications/x3dh/
228	[TLS]	[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999. http://www.ietf.org/rfc/rfc2246.txt , superseded by [RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006. http://www.ietf.org/rfc/rfc4346.txt , which was superseded by [5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008. URL: http://www.ietf.org/rfc/rfc5246.txt
229	[TLS12]	[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008. URL: http://www.ietf.org/rfc/rfc5246.txt
230	[TLS13]	[RFC8446] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018. URL: http://www.ietf.org/rfc/rfc8446.txt
231	[WIM]	WAP. Wireless Identity Module. — WAP-260-WIM-20010712-a. July 2001. URL: http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.aspx?DocName=/wap/wap-260-wim-20010712-a.pdf
232	[WPKI]	Wireless Application Protocol: Public Key Infrastructure Definition. — WAP-217-WPKI-20010424-a. April 2001. URL: http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.aspx?DocName=/wap/wap-217-wpki-20010424-a.pdf
233	[WTLS]	WAP. Wireless Transport Layer Security Version — WAP-261-WTLS-20010406-a. April 2001. URL: http://technical.openmobilealliance.org/tech/affiliates/LicenseAgreement.aspx?DocName=/wap/wap-261-wtls-20010406-a.pdf
234	[XEDDSA]	The XEdDSA and VXEddSA Signature Schemes - Revision 1, 2016-10-20, Trevor Perrin (editor) URL: https://signal.org/docs/specifications/xeddsa/
235	[X.500]	ITU-T. Information Technology — Open Systems Interconnection — The Directory: Overview of Concepts, Models and Services. February 2001. Identical to ISO/IEC 9594-1
236	[X.509]	ITU-T. Information Technology — Open Systems Interconnection — The Directory: Public-key and Attribute Certificate Frameworks. March 2000. Identical to ISO/IEC 9594-8
237	[X.680]	ITU-T. Information Technology — Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. July 2002. Identical to ISO/IEC 8824-1
238	[X.690]	ITU-T. Information Technology — ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER). July 2002. Identical to ISO/IEC 8825-1
239		
240		
241		
242		
243		
244		
245		
246		
247		
248		
249		
250		
251		
252		
253		
254		
255		
256		
257		
258		
259		
260		
261		
262		
263		
264		
265		
266		
267		
268		
269		
270		
271		
272		
273		
274		
275		

276 2 Mechanisms

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

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

286 Each mechanism description is preceded by a table, of the following format, mapping mechanisms to
287 API functions.

288

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive

289 1 SR = SignRecover, VR = VerifyRecover.

290 2 Single-part operations only.

291 3 Mechanism can only be used for wrapping, not unwrapping.

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

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

295 2.1 RSA

296 *Table 1, Mechanisms vs. Functions*

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_RSA_PKCS_KEY_PAIR_GEN					✓		
CKM_RSA_X9_31_KEY_PAIR_GEN					✓		
CKM_RSA_PKCS	✓ ²	✓ ²	✓			✓	
CKM_RSA_PKCS_OAEP	✓ ²					✓	
CKM_RSA_PKCS_PSS		✓ ²					
CKM_RSA_9796		✓ ²	✓				
CKM_RSA_X_509	✓ ²	✓ ²	✓			✓	
CKM_RSA_X9_31		✓ ²					
CKM_SHA1_RSA_PKCS		✓					
CKM_SHA256_RSA_PKCS		✓					
CKM_SHA384_RSA_PKCS		✓					

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ₁	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_SHA512_RSA_PKCS		✓					
CKM_SHA1_RSA_PKCS_PSS		✓					
CKM_SHA256_RSA_PKCS_PSS		✓					
CKM_SHA384_RSA_PKCS_PSS		✓					
CKM_SHA512_RSA_PKCS_PSS		✓					
CKM_SHA1_RSA_X9_31		✓					
CKM_RSA_PKCS TPM_1_1	✓ ²					✓	
CKM_RSA_PKCS_OAEP TPM_1_1	✓ ²					✓	
CKM_SHA3_224_RSA_PKCS		✓					
CKM_SHA3_256_RSA_PKCS		✓					
CKM_SHA3_384_RSA_PKCS		✓					
CKM_SHA3_512_RSA_PKCS		✓					
CKM_SHA3_224_RSA_PKCS_PSS		✓					
CKM_SHA3_256_RSA_PKCS_PSS		✓					
CKM_SHA3_384_RSA_PKCS_PSS		✓					
CKM_SHA3_512_RSA_PKCS_PSS		✓					

297 2.1.1 Definitions

298 This section defines the RSA key type "CKK_RSA" for type CK_KEY_TYPE as used in the
 299 CKA_KEY_TYPE attribute of RSA key objects.

300 Mechanisms:

301 CKM_RSA_PKCS_KEY_PAIR_GEN
 302 CKM_RSA_PKCS
 303 CKM_RSA_9796
 304 CKM_RSA_X_509
 305 CKM_MD2_RSA_PKCS
 306 CKM_MD5_RSA_PKCS
 307 CKM_SHA1_RSA_PKCS
 308 CKM_SHA224_RSA_PKCS
 309 CKM_SHA256_RSA_PKCS
 310 CKM_SHA384_RSA_PKCS
 311 CKM_SHA512_RSA_PKCS
 312 CKM_RIPEMD128_RSA_PKCS
 313 CKM_RIPEMD160_RSA_PKCS
 314 CKM_RSA_PKCS_OAEP
 315 CKM_RSA_X9_31_KEY_PAIR_GEN
 316 CKM_RSA_X9_31
 317 CKM_SHA1_RSA_X9_31
 318 CKM_RSA_PKCS_PSS
 319 CKM_SHA1_RSA_PKCS_PSS

```

320      CKM_SHA224_RSA_PKCS_PSS
321      CKM_SHA256_RSA_PKCS_PSS
322      CKM_SHA512_RSA_PKCS_PSS
323      CKM_SHA384_RSA_PKCS_PSS
324      CKM_RSA_PKCS TPM_1_1
325      CKM_RSA_PKCS_OAEP TPM_1_1
326      CKM_RSA_AES_KEY_WRAP
327      CKM_SHA3_224_RSA_PKCS
328      CKM_SHA3_256_RSA_PKCS
329      CKM_SHA3_384_RSA_PKCS
330      CKM_SHA3_512_RSA_PKCS
331      CKM_SHA3_224_RSA_PKCS_PSS
332      CKM_SHA3_256_RSA_PKCS_PSS
333      CKM_SHA3_384_RSA_PKCS_PSS
334      CKM_SHA3_512_RSA_PKCS_PSS
335

```

336 2.1.2 RSA public key objects

337 RSA public key objects (object class **CKO_PUBLIC_KEY**, key type **CKK_RSA**) hold RSA public keys.
 338 The following table defines the RSA public key object attributes, in addition to the common attributes
 339 defined for this object class:

340 *Table 2, RSA Public Key Object Attributes*

Attribute	Data type	Meaning
CKA_MODULUS ^{1,4}	Big integer	Modulus n
CKA_MODULUS_BITS ^{2,3}	CK ULONG	Length in bits of modulus n
CKA_PUBLIC_EXPONENT ¹	Big integer	Public exponent e

341 - Refer to [PKCS11-Base] table 11 for footnotes

342 Depending on the token, there may be limits on the length of key components. See PKCS #1 for more
 343 information on RSA keys.

344 The following is a sample template for creating an RSA public key object:

```

345     CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
346     CK_KEY_TYPE keyType = CKK_RSA;
347     CK_UTF8CHAR label[] = "An RSA public key object";
348     CK_BYTE modulus[] = {...};
349     CK_BYTE exponent[] = {...};
350     CK_BBOOL true = CK_TRUE;
351     CK_ATTRIBUTE template[] = {
352         {CKA_CLASS, &class, sizeof(class)},
353         {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
354         {CKA_TOKEN, &true, sizeof(true)},
355         {CKA_LABEL, label, sizeof(label)-1},
356         {CKA_WRAP, &true, sizeof(true)},
357         {CKA_ENCRYPT, &true, sizeof(true)},
358         {CKA_MODULUS, modulus, sizeof(modulus)},

```

```
359     { CKA_PUBLIC_EXPONENT, exponent, sizeof(exponent) }  
360 }
```

361 2.1.3 RSA private key objects

362 RSA private key objects (object class **CKO_PRIVATE_KEY**, key type **CKK_RSA**) hold RSA private keys.
363 The following table defines the RSA private key object attributes, in addition to the common attributes
364 defined for this object class:

365 *Table 3, RSA Private Key Object Attributes*

Attribute	Data type	Meaning
CKA_MODULUS ^{1,4,6}	Big integer	Modulus n
CKA_PUBLIC_EXPONENT ^{4,6}	Big integer	Public exponent e
CKA_PRIVATE_EXPONENT ^{1,4,6,7}	Big integer	Private exponent d
CKA_PRIME_1 ^{4,6,7}	Big integer	Prime p
CKA_PRIME_2 ^{4,6,7}	Big integer	Prime q
CKA_EXPONENT_1 ^{4,6,7}	Big integer	Private exponent d modulo $p-1$
CKA_EXPONENT_2 ^{4,6,7}	Big integer	Private exponent d modulo $q-1$
CKA_COEFFICIENT ^{4,6,7}	Big integer	CRT coefficient $q^{-1} \bmod p$

366 - Refer to [PKCS11-Base] table 11 for footnotes

367 Depending on the token, there may be limits on the length of the key components. See PKCS #1 for
368 more information on RSA keys.

369 Tokens vary in what they actually store for RSA private keys. Some tokens store all of the above
370 attributes, which can assist in performing rapid RSA computations. Other tokens might store only the
371 **CKA_MODULUS** and **CKA_PRIVATE_EXPONENT** values. Effective with version 2.40, tokens MUST
372 also store **CKA_PUBLIC_EXPONENT**. This permits the retrieval of sufficient data to reconstitute the
373 associated public key.

374 Because of this, Cryptoki is flexible in dealing with RSA private key objects. When a token generates an
375 RSA private key, it stores whichever of the fields in Table 3 it keeps track of. Later, if an application asks
376 for the values of the key's various attributes, Cryptoki supplies values only for attributes whose values it
377 can obtain (i.e., if Cryptoki is asked for the value of an attribute it cannot obtain, the request fails). Note
378 that a Cryptoki implementation may or may not be able and/or willing to supply various attributes of RSA
379 private keys which are not actually stored on the token. *E.g.*, if a particular token stores values only for
380 the **CKA_PRIVATE_EXPONENT**, **CKA_PRIME_1**, and **CKA_PRIME_2** attributes, then Cryptoki is
381 certainly *able* to report values for all the attributes above (since they can all be computed efficiently from
382 these three values). However, a Cryptoki implementation may or may not actually do this extra
383 computation. The only attributes from Table 3 for which a Cryptoki implementation is *required* to be able
384 to return values are **CKA_MODULUS** and **CKA_PRIVATE_EXPONENT**.

385 If an RSA private key object is created on a token, and more attributes from Table 3 are supplied to the
386 object creation call than are supported by the token, the extra attributes are likely to be thrown away. If
387 an attempt is made to create an RSA private key object on a token with insufficient attributes for that
388 particular token, then the object creation call fails and returns **CKR_TEMPLATE_INCOMPLETE**.

389 Note that when generating an RSA private key, there is no **CKA_MODULUS_BITS** attribute specified.
390 This is because RSA private keys are only generated as part of an RSA key pair, and the
391 **CKA_MODULUS_BITS** attribute for the pair is specified in the template for the RSA public key.

392 The following is a sample template for creating an RSA private key object:

```
393 CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;  
394 CK_KEY_TYPE keyType = CKK_RSA;  
395 CK_UTF8CHAR label[] = "An RSA private key object";  
396 CK_BYTE subject[] = {...};  
397 CK_BYTE id[] = {123};
```

```

398     CK_BYTE modulus[] = {...};
399     CK_BYTE publicExponent[] = {...};
400     CK_BYTE privateExponent[] = {...};
401     CK_BYTE prime1[] = {...};
402     CK_BYTE prime2[] = {...};
403     CK_BYTE exponent1[] = {...};
404     CK_BYTE exponent2[] = {...};
405     CK_BYTE coefficient[] = {...};
406     CK_BBOOL true = CK_TRUE;
407     CK_ATTRIBUTE template[] = {
408         {CKA_CLASS, &class, sizeof(class)},
409         {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
410         {CKA_TOKEN, &true, sizeof(true)},
411         {CKA_LABEL, label, sizeof(label)-1},
412         {CKA SUBJECT, subject, sizeof(subject)},
413         {CKA_ID, id, sizeof(id)},
414         {CKA_SENSITIVE, &true, sizeof(true)},
415         {CKA_DECRYPT, &true, sizeof(true)},
416         {CKA_SIGN, &true, sizeof(true)},
417         {CKA_MODULUS, modulus, sizeof(modulus)},
418         {CKA_PUBLIC_EXPONENT, publicExponent,
419             sizeof(publicExponent)},
420         {CKA_PRIVATE_EXPONENT, privateExponent,
421             sizeof(privateExponent)},
422         {CKA_PRIME_1, prime1, sizeof(prime1)},
423         {CKA_PRIME_2, prime2, sizeof(prime2)},
424         {CKA_EXPONENT_1, exponent1, sizeof(exponent1)},
425         {CKA_EXPONENT_2, exponent2, sizeof(exponent2)},
426         {CKA_COEFFICIENT, coefficient, sizeof(coefficient)}}
427     };

```

2.1.4 PKCS #1 RSA key pair generation

429 The PKCS #1 RSA key pair generation mechanism, denoted **CKM_RSA_PKCS_KEY_PAIR_GEN**, is a
430 key pair generation mechanism based on the RSA public-key cryptosystem, as defined in PKCS #1.

431 It does not have a parameter.

432 The mechanism generates RSA public/private key pairs with a particular modulus length in bits and public
433 exponent, as specified in the **CKA_MODULUS_BITS** and **CKA_PUBLIC_EXPONENT** attributes of the
434 template for the public key. The **CKA_PUBLIC_EXPONENT** may be omitted in which case the
435 mechanism shall supply the public exponent attribute using the default value of 0x10001 (65537).
436 Specific implementations may use a random value or an alternative default if 0x10001 cannot be used by
437 the token.

438 Note: Implementations strictly compliant with version 2.11 or prior versions may generate an error
439 if this attribute is omitted from the template. Experience has shown that many implementations of 2.11
440 and prior did allow the **CKA_PUBLIC_EXPONENT** attribute to be omitted from the template, and
441 behaved as described above. The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**,
442 **CKA_MODULUS**, and **CKA_PUBLIC_EXPONENT** attributes to the new public key.
443 **CKA_PUBLIC_EXPONENT** will be copied from the template if supplied.
444 **CKR_TEMPLATE_INCONSISTENT** shall be returned if the implementation cannot use the supplied
445 exponent value. It contributes the **CKA_CLASS** and **CKA_KEY_TYPE** attributes to the new private key; it

446 may also contribute some of the following attributes to the new private key: **CKA_MODULUS**,
447 **CKA_PUBLIC_EXPONENT**, **CKA_PRIVATE_EXPONENT**, **CKA_PRIME_1**, **CKA_PRIME_2**,
448 **CKA_EXPONENT_1**, **CKA_EXPONENT_2**, **CKA_COEFFICIENT**. Other attributes supported by the
449 RSA public and private key types (specifically, the flags indicating which functions the keys support) may
450 also be specified in the templates for the keys, or else are assigned default initial values.
451 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
452 specify the supported range of RSA modulus sizes, in bits.

453 2.1.5 X9.31 RSA key pair generation

454 The X9.31 RSA key pair generation mechanism, denoted **CKM_RSA_X9_31_KEY_PAIR_GEN**, is a key
455 pair generation mechanism based on the RSA public-key cryptosystem, as defined in X9.31.

456 It does not have a parameter.

457 The mechanism generates RSA public/private key pairs with a particular modulus length in bits and public
458 exponent, as specified in the **CKA_MODULUS_BITS** and **CKA_PUBLIC_EXPONENT** attributes of the
459 template for the public key.

460 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_MODULUS**, and
461 **CKA_PUBLIC_EXPONENT** attributes to the new public key. It contributes the **CKA_CLASS** and
462 **CKA_KEY_TYPE** attributes to the new private key; it may also contribute some of the following attributes
463 to the new private key: **CKA_MODULUS**, **CKA_PUBLIC_EXPONENT**, **CKA_PRIVATE_EXPONENT**,
464 **CKA_PRIME_1**, **CKA_PRIME_2**, **CKA_EXPONENT_1**, **CKA_EXPONENT_2**, **CKA_COEFFICIENT**.
465 Other attributes supported by the RSA public and private key types (specifically, the flags indicating which
466 functions the keys support) may also be specified in the templates for the keys, or else are assigned
467 default initial values. Unlike the **CKM_RSA_PKCS_KEY_PAIR_GEN** mechanism, this mechanism is
468 guaranteed to generate *p* and *q* values, **CKA_PRIME_1** and **CKA_PRIME_2** respectively, that meet the
469 strong primes requirement of X9.31.

470 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
471 specify the supported range of RSA modulus sizes, in bits.

472 2.1.6 PKCS #1 v1.5 RSA

473 The PKCS #1 v1.5 RSA mechanism, denoted **CKM_RSA_PKCS**, is a multi-purpose mechanism based
474 on the RSA public-key cryptosystem and the block formats initially defined in PKCS #1 v1.5. It supports
475 single-part encryption and decryption; single-part signatures and verification with and without message
476 recovery; key wrapping; and key unwrapping. This mechanism corresponds only to the part of PKCS #1
477 v1.5 that involves RSA; it does not compute a message digest or a DigestInfo encoding as specified for
478 the md2withRSAEncryption and md5withRSAEncryption algorithms in PKCS #1 v1.5 .

479 This mechanism does not have a parameter.

480 This mechanism can wrap and unwrap any secret key of appropriate length. Of course, a particular token
481 may not be able to wrap/unwrap every appropriate-length secret key that it supports. For wrapping, the
482 “input” to the encryption operation is the value of the **CKA_VALUE** attribute of the key that is wrapped;
483 similarly for unwrapping. The mechanism does not wrap the key type or any other information about the
484 key, except the key length; the application must convey these separately. In particular, the mechanism
485 contributes only the **CKA_CLASS** and **CKA_VALUE** (and **CKA_VALUE_LEN**, if the key has it) attributes
486 to the recovered key during unwrapping; other attributes must be specified in the template.

487 Constraints on key types and the length of the data are summarized in the following table. For
488 encryption, decryption, signatures and signature verification, the input and output data may begin at the
489 same location in memory. In the table, *k* is the length in bytes of the RSA modulus.

490 Table 4, PKCS #1 v1.5 RSA: Key And Data Length

Function	Key type	Input length	Output length	Comments
C_Encrypt ¹	RSA public key	$\leq k-11$	k	block type 02
C_Decrypt ¹	RSA private key	k	$\leq k-11$	block type 02
C_Sign ¹	RSA private key	$\leq k-11$	k	block type 01
C_SignRecover	RSA private key	$\leq k-11$	k	block type 01
C_Verify ¹	RSA public key	$\leq k-11, k^2$	N/A	block type 01
C_VerifyRecover	RSA public key	k	$\leq k-11$	block type 01
C_WrapKey	RSA public key	$\leq k-11$	k	block type 02
C_UnwrapKey	RSA private key	k	$\leq k-11$	block type 02

491 ¹ Single-part operations only.492 ² Data length, signature length.493 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
494 specify the supported range of RSA modulus sizes, in bits.495

2.1.7 PKCS #1 RSA OAEP mechanism parameters

496

- ◆ **CK_RSA_PKCS_MGF_TYPE; CK_RSA_PKCS_MGF_TYPE_PTR**

497 **CK_RSA_PKCS_MGF_TYPE** is used to indicate the Message Generation Function (MGF) applied to a
498 message block when formatting a message block for the PKCS #1 OAEP encryption scheme or the
499 PKCS #1 PSS signature scheme. It is defined as follows:500

```
typedef CK ULONG CK_RSA_PKCS_MGF_TYPE;
```

501

502 The following MGFs are defined in PKCS #1. The following table lists the defined functions.

503 *Table 5, PKCS #1 Mask Generation Functions*

Source Identifier	Value
CKG_MGF1_SHA1	0x00000001UL
CKG_MGF1_SHA224	0x00000005UL
CKG_MGF1_SHA256	0x00000002UL
CKG_MGF1_SHA384	0x00000003UL
CKG_MGF1_SHA512	0x00000004UL
CKG_MGF1_SHA3_224	0x00000006UL
CKG_MGF1_SHA3_256	0x00000007UL
CKG_MGF1_SHA3_384	0x00000008UL
CKG_MGF1_SHA3_512	0x00000009UL

504 **CK_RSA_PKCS_MGF_TYPE_PTR** is a pointer to a **CK_RSA_PKCS_MGF_TYPE**.505

- ◆ **CK_RSA_PKCS_OAEP_SOURCE_TYPE;**
CK_RSA_PKCS_OAEP_SOURCE_TYPE_PTR

507 **CK_RSA_PKCS_OAEP_SOURCE_TYPE** is used to indicate the source of the encoding parameter
508 when formatting a message block for the PKCS #1 OAEP encryption scheme. It is defined as follows:509

```
typedef CK ULONG CK_RSA_PKCS_OAEP_SOURCE_TYPE;
```

510
511 The following encoding parameter sources are defined in PKCS #1. The following table lists the defined
512 sources along with the corresponding data type for the *pSourceData* field in the
513 **CK_RSA_PKCS_OAEP_PARAMS** structure defined below.

514 *Table 6, PKCS #1 RSA OAEP: Encoding parameter sources*

Source Identifier	Value	Data Type
CKZ_DATA_SPECIFIED	0x00000001UL	Array of CK_BYTE containing the value of the encoding parameter. If the parameter is empty, <i>pSourceData</i> must be NULL and <i>ulSourceDataLen</i> must be zero.

515 **CK_RSA_PKCS_OAEP_SOURCE_TYPE_PTR** is a pointer to a
516 **CK_RSA_PKCS_OAEP_SOURCE_TYPE**.

517 ◆ **CK_RSA_PKCS_OAEP_PARAMS; CK_RSA_PKCS_OAEP_PARAMS_PTR**

518 **CK_RSA_PKCS_OAEP_PARAMS** is a structure that provides the parameters to the
519 **CKM_RSA_PKCS_OAEP** mechanism. The structure is defined as follows:

```
520     typedef struct CK_RSA_PKCS_OAEP_PARAMS {  
521         CK_MECHANISM_TYPE           hashAlg;  
522         CK_RSA_PKCS_MGF_TYPE        mgf;  
523         CK_RSA_PKCS_OAEP_SOURCE_TYPE source;  
524         CK_VOID_PTR                pSourceData;  
525         CK_ULONG                   ulSourceDataLen;  
526     } CK_RSA_PKCS_OAEP_PARAMS;
```

527
528 The fields of the structure have the following meanings:

529 *hashAlg* *mechanism ID of the message digest algorithm used to calculate*
530 *the digest of the encoding parameter*

531 *mgf* *mask generation function to use on the encoded block*

532 *source* *source of the encoding parameter*

533 *pSourceData* *data used as the input for the encoding parameter source*

534 *ulSourceDataLen* *length of the encoding parameter source input*

535 **CK_RSA_PKCS_OAEP_PARAMS_PTR** is a pointer to a **CK_RSA_PKCS_OAEP_PARAMS**.

536

537 **2.1.8 PKCS #1 RSA OAEP**

538 The PKCS #1 RSA OAEP mechanism, denoted **CKM_RSA_PKCS_OAEP**, is a multi-purpose
539 mechanism based on the RSA public-key cryptosystem and the OAEP block format defined in PKCS #1.
540 It supports single-part encryption and decryption; key wrapping; and key unwrapping.

541 It has a parameter, a **CK_RSA_PKCS_OAEP_PARAMS** structure.

542 This mechanism can wrap and unwrap any secret key of appropriate length. Of course, a particular token
543 may not be able to wrap/unwrap every appropriate-length secret key that it supports. For wrapping, the
544 “input” to the encryption operation is the value of the **CKA_VALUE** attribute of the key that is wrapped;
545 similarly for unwrapping. The mechanism does not wrap the key type or any other information about the

546 key, except the key length; the application must convey these separately. In particular, the mechanism
547 contributes only the **CKA_CLASS** and **CKA_VALUE** (and **CKA_VALUE_LEN**, if the key has it) attributes
548 to the recovered key during unwrapping; other attributes must be specified in the template.

549 Constraints on key types and the length of the data are summarized in the following table. For encryption
550 and decryption, the input and output data may begin at the same location in memory. In the table, k is the
551 length in bytes of the RSA modulus, and $hLen$ is the output length of the message digest algorithm
552 specified by the *hashAlg* field of the **CK_RSA_PKCS_OAEP_PARAMS** structure.

553 *Table 7, PKCS #1 RSA OAEP: Key And Data Length*

Function	Key type	Input length	Output length
C_Encrypt ¹	RSA public key	$\leq k-2-2hLen$	k
C_Decrypt ¹	RSA private key	k	$\leq k-2-2hLen$
C_WrapKey	RSA public key	$\leq k-2-2hLen$	k
C_UnwrapKey	RSA private key	k	$\leq k-2-2hLen$

554 ¹ Single-part operations only.

555 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
556 specify the supported range of RSA modulus sizes, in bits.

557 2.1.9 PKCS #1 RSA PSS mechanism parameters

558 ♦ **CK_RSA_PKCS_PSS_PARAMS; CK_RSA_PKCS_PSS_PARAMS_PTR**

559 **CK_RSA_PKCS_PSS_PARAMS** is a structure that provides the parameters to the
560 **CKM_RSA_PKCS_PSS** mechanism. The structure is defined as follows:

```
561     typedef struct CK_RSA_PKCS_PSS_PARAMS {  
562         CK_MECHANISM_TYPE      hashAlg;  
563         CK_RSA_PKCS_MGF_TYPE   mgf;  
564         CK ULONG               sLen;  
565     } CK_RSA_PKCS_PSS_PARAMS;
```

566

567 The fields of the structure have the following meanings:

568 *hashAlg* *hash algorithm used in the PSS encoding; if the signature
569 mechanism does not include message hashing, then this value must
570 be the mechanism used by the application to generate the message
571 hash; if the signature mechanism includes hashing, then this value
572 must match the hash algorithm indicated by the signature
573 mechanism*

574 *mgf* *mask generation function to use on the encoded block*

575 *sLen* *length, in bytes, of the salt value used in the PSS encoding; typical
576 values are the length of the message hash and zero*

577 **CK_RSA_PKCS_PSS_PARAMS_PTR** is a pointer to a **CK_RSA_PKCS_PSS_PARAMS**.

578 2.1.10 PKCS #1 RSA PSS

579 The PKCS #1 RSA PSS mechanism, denoted **CKM_RSA_PKCS_PSS**, is a mechanism based on the
580 RSA public-key cryptosystem and the PSS block format defined in PKCS #1. It supports single-part
581 signature generation and verification without message recovery. This mechanism corresponds only to the

582 part of PKCS #1 that involves block formatting and RSA, given a hash value; it does not compute a hash
583 value on the message to be signed.

584 It has a parameter, a **CK_RSA_PKCS_PSS_PARAMS** structure. The *sLen* field must be less than or
585 equal to $k^* - 2 \cdot hLen$ and *hLen* is the length of the input to the *C_Sign* or *C_Verify* function. k^* is the length
586 in bytes of the RSA modulus, except if the length in bits of the RSA modulus is one more than a multiple
587 of 8, in which case k^* is one less than the length in bytes of the RSA modulus.

588 Constraints on key types and the length of the data are summarized in the following table. In the table, *k*
589 is the length in bytes of the RSA.

590 *Table 8, PKCS #1 RSA PSS: Key And Data Length*

Function	Key type	Input length	Output length
<i>C_Sign</i> ¹	RSA private key	<i>hLen</i>	<i>k</i>
<i>C_Verify</i> ¹	RSA public key	<i>hLen, k</i>	N/A

591 ¹ Single-part operations only.

592 ² Data length, signature length.

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

595 2.1.11 ISO/IEC 9796 RSA

596 The ISO/IEC 9796 RSA mechanism, denoted **CKM_RSA_9796**, is a mechanism for single-part
597 signatures and verification with and without message recovery based on the RSA public-key
598 cryptosystem and the block formats defined in ISO/IEC 9796 and its annex A.

599 This mechanism processes only byte strings, whereas ISO/IEC 9796 operates on bit strings. Accordingly,
600 the following transformations are performed:

- 601 • Data is converted between byte and bit string formats by interpreting the most-significant bit of the
602 leading byte of the byte string as the leftmost bit of the bit string, and the least-significant bit of the
603 trailing byte of the byte string as the rightmost bit of the bit string (this assumes the length in bits of
604 the data is a multiple of 8).
- 605 • A signature is converted from a bit string to a byte string by padding the bit string on the left with 0 to
606 7 zero bits so that the resulting length in bits is a multiple of 8, and converting the resulting bit string
607 as above; it is converted from a byte string to a bit string by converting the byte string as above, and
608 removing bits from the left so that the resulting length in bits is the same as that of the RSA modulus.

609 This mechanism does not have a parameter.

610 Constraints on key types and the length of input and output data are summarized in the following table.
611 In the table, *k* is the length in bytes of the RSA modulus.

612 *Table 9, ISO/IEC 9796 RSA: Key And Data Length*

Function	Key type	Input length	Output length
<i>C_Sign</i> ¹	RSA private key	$\leq \lfloor k/2 \rfloor$	<i>k</i>
<i>C_SignRecover</i>	RSA private key	$\leq \lfloor k/2 \rfloor$	<i>k</i>
<i>C_Verify</i> ¹	RSA public key	$\leq \lfloor k/2 \rfloor, k^2$	N/A
<i>C_VerifyRecover</i>	RSA public key	<i>k</i>	$\leq \lfloor k/2 \rfloor$

613 ¹ Single-part operations only.

614 ² Data length, signature length.

615 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
616 specify the supported range of RSA modulus sizes, in bits.

617 2.1.12 X.509 (raw) RSA

618 The X.509 (raw) RSA mechanism, denoted **CKM_RSA_X_509**, is a multi-purpose mechanism based on
619 the RSA public-key cryptosystem. It supports single-part encryption and decryption; single-part signatures
620 and verification with and without message recovery; key wrapping; and key unwrapping. All these
621 operations are based on so-called “raw” RSA, as assumed in X.509.

622 “Raw” RSA as defined here encrypts a byte string by converting it to an integer, most-significant byte first,
623 applying “raw” RSA exponentiation, and converting the result to a byte string, most-significant byte first.
624 The input string, considered as an integer, must be less than the modulus; the output string is also less
625 than the modulus.

626 This mechanism does not have a parameter.

627 This mechanism can wrap and unwrap any secret key of appropriate length. Of course, a particular token
628 may not be able to wrap/unwrap every appropriate-length secret key that it supports. For wrapping, the
629 “input” to the encryption operation is the value of the **CKA_VALUE** attribute of the key that is wrapped;
630 similarly for unwrapping. The mechanism does not wrap the key type, key length, or any other
631 information about the key; the application must convey these separately, and supply them when
632 unwrapping the key.

633 Unfortunately, X.509 does not specify how to perform padding for RSA encryption. For this mechanism,
634 padding should be performed by prepending plaintext data with 0-valued bytes. In effect, to encrypt the
635 sequence of plaintext bytes $b_1 b_2 \dots b_n$ ($n \leq k$), Cryptoki forms $P=2^{n-1}b_1+2^{n-2}b_2+\dots+b_n$. This number must
636 be less than the RSA modulus. The k -byte ciphertext (k is the length in bytes of the RSA modulus) is
637 produced by raising P to the RSA public exponent modulo the RSA modulus. Decryption of a k -byte
638 ciphertext C is accomplished by raising C to the RSA private exponent modulo the RSA modulus, and
639 returning the resulting value as a sequence of exactly k bytes. If the resulting plaintext is to be used to
640 produce an unwrapped key, then however many bytes are specified in the template for the length of the
641 key are taken from the end of this sequence of bytes.

642 Technically, the above procedures may differ very slightly from certain details of what is specified in
643 X.509.

644 Executing cryptographic operations using this mechanism can result in the error returns
645 **CKR_DATA_INVALID** (if plaintext is supplied which has the same length as the RSA modulus and is
646 numerically at least as large as the modulus) and **CKR_ENCRYPTED_DATA_INVALID** (if ciphertext is
647 supplied which has the same length as the RSA modulus and is numerically at least as large as the
648 modulus).

649 Constraints on key types and the length of input and output data are summarized in the following table.
650 In the table, k is the length in bytes of the RSA modulus.

651 *Table 10, X.509 (Raw) RSA: Key And Data Length*

Function	Key type	Input length	Output length
C_Encrypt¹	RSA public key	$\leq k$	k
C_Decrypt¹	RSA private key	k	k
C_Sign¹	RSA private key	$\leq k$	k
C_SignRecover	RSA private key	$\leq k$	k
C_Verify¹	RSA public key	$\leq k, k^2$	N/A
C_VerifyRecover	RSA public key	k	k
C_WrapKey	RSA public key	$\leq k$	k
C_UnwrapKey	RSA private key	k	$\leq k$ (specified in template)

652 ¹ Single-part operations only.

653 ² Data length, signature length.

654 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
655 specify the supported range of RSA modulus sizes, in bits.

656 This mechanism is intended for compatibility with applications that do not follow the PKCS #1 or ISO/IEC
657 9796 block formats.

658 2.1.13 ANSI X9.31 RSA

659 The ANSI X9.31 RSA mechanism, denoted **CKM_RSA_X9_31**, is a mechanism for single-part signatures
660 and verification without message recovery based on the RSA public-key cryptosystem and the block
661 formats defined in ANSI X9.31.

662 This mechanism applies the header and padding fields of the hash encapsulation. The trailer field must
663 be applied by the application.

664 This mechanism processes only byte strings, whereas ANSI X9.31 operates on bit strings. Accordingly,
665 the following transformations are performed:

- 666 • Data is converted between byte and bit string formats by interpreting the most-significant bit of the
667 leading byte of the byte string as the leftmost bit of the bit string, and the least-significant bit of the
668 trailing byte of the byte string as the rightmost bit of the bit string (this assumes the length in bits of
669 the data is a multiple of 8).
- 670 • A signature is converted from a bit string to a byte string by padding the bit string on the left with 0 to
671 7 zero bits so that the resulting length in bits is a multiple of 8, and converting the resulting bit string
672 as above; it is converted from a byte string to a bit string by converting the byte string as above, and
673 removing bits from the left so that the resulting length in bits is the same as that of the RSA modulus.

674 This mechanism does not have a parameter.

675 Constraints on key types and the length of input and output data are summarized in the following table.
676 In the table, k is the length in bytes of the RSA modulus. For all operations, the k value must be at least
677 128 and a multiple of 32 as specified in ANSI X9.31.

678 *Table 11, ANSI X9.31 RSA: Key And Data Length*

Function	Key type	Input length	Output length
C_Sign ¹	RSA private key	$\leq k-2$	k
C_Verify ¹	RSA public key	$\leq k-2, k^2$	N/A

679 ¹ Single-part operations only.

680 ² Data length, signature length.

681 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
682 specify the supported range of RSA modulus sizes, in bits.

683 2.1.14 PKCS #1 v1.5 RSA signature with MD2, MD5, SHA-1, SHA-256, SHA- 684 384, SHA-512, RIPE-MD 128 or RIPE-MD 160

685 The PKCS #1 v1.5 RSA signature with MD2 mechanism, denoted **CKM_MD2_RSA_PKCS**, performs
686 single- and multiple-part digital signatures and verification operations without message recovery. The
687 operations performed are as described initially in PKCS #1 v1.5 with the object identifier
688 md2WithRSAEncryption, and as in the scheme RSASSA-PKCS1-v1_5 in the current version of PKCS #1,
689 where the underlying hash function is MD2.

690 Similarly, the PKCS #1 v1.5 RSA signature with MD5 mechanism, denoted **CKM_MD5_RSA_PKCS**,
691 performs the same operations described in PKCS #1 with the object identifier md5WithRSAEncryption.
692 The PKCS #1 v1.5 RSA signature with SHA-1 mechanism, denoted **CKM_SHA1_RSA_PKCS**, performs
693 the same operations, except that it uses the hash function SHA-1 with object identifier
694 sha1WithRSAEncryption.

695 Likewise, the PKCS #1 v1.5 RSA signature with SHA-256, SHA-384, and SHA-512 mechanisms, denoted
696 **CKM_SHA256_RSA_PKCS**, **CKM_SHA384_RSA_PKCS**, and **CKM_SHA512_RSA_PKCS** respectively,
697 perform the same operations using the SHA-256, SHA-384 and SHA-512 hash functions with the object

698 identifiers sha256WithRSAEncryption, sha384WithRSAEncryption and sha512WithRSAEncryption
 699 respectively.
 700 The PKCS #1 v1.5 RSA signature with RIPEMD-128 or RIPEMD-160, denoted
 701 **CKM_RIPEMD128_RSA_PKCS** and **CKM_RIPEMD160_RSA_PKCS** respectively, perform the same
 702 operations using the RIPE-MD 128 and RIPE-MD 160 hash functions.
 703 None of these mechanisms has a parameter.
 704 Constraints on key types and the length of the data for these mechanisms are summarized in the
 705 following table. In the table, k is the length in bytes of the RSA modulus. For the PKCS #1 v1.5 RSA
 706 signature with MD2 and PKCS #1 v1.5 RSA signature with MD5 mechanisms, k must be at least 27; for
 707 the PKCS #1 v1.5 RSA signature with SHA-1 mechanism, k must be at least 31, and so on for other
 708 underlying hash functions, where the minimum is always 11 bytes more than the length of the hash value.
 709 *Table 12, PKCS #1 v1.5 RSA Signatures with Various Hash Functions: Key And Data Length*

Function	Key type	Input length	Output length	Comments
C_Sign	RSA private key	any	k	block type 01
C_Verify	RSA public key	any, k^2	N/A	block type 01

710 2 Data length, signature length.
 711 For these mechanisms, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO**
 712 structure specify the supported range of RSA modulus sizes, in bits.

2.1.15 PKCS #1 v1.5 RSA signature with SHA-224

714 The PKCS #1 v1.5 RSA signature with SHA-224 mechanism, denoted **CKM_SHA224_RSA_PKCS**,
 715 performs similarly as the other **CKM_SHAX_RSA_PKCS** mechanisms but uses the SHA-224 hash
 716 function.

2.1.16 PKCS #1 RSA PSS signature with SHA-224

718 The PKCS #1 RSA PSS signature with SHA-224 mechanism, denoted **CKM_SHA224_RSA_PKCS_PSS**,
 719 performs similarly as the other **CKM_SHAX_RSA_PKCS_PSS** mechanisms but uses the SHA-224 hash
 720 function.

2.1.17 PKCS #1 RSA PSS signature with SHA-1, SHA-256, SHA-384 or SHA-512

723 The PKCS #1 RSA PSS signature with SHA-1 mechanism, denoted **CKM_SHA1_RSA_PKCS_PSS**,
 724 performs single- and multiple-part digital signatures and verification operations without message
 725 recovery. The operations performed are as described in PKCS #1 with the object identifier id-RSASSA-
 726 PSS, i.e., as in the scheme RSASSA-PSS in PKCS #1 where the underlying hash function is SHA-1.
 727 The PKCS #1 RSA PSS signature with SHA-256, SHA-384, and SHA-512 mechanisms, denoted
 728 **CKM_SHA256_RSA_PKCS_PSS**, **CKM_SHA384_RSA_PKCS_PSS**, and
 729 **CKM_SHA512_RSA_PKCS_PSS** respectively, perform the same operations using the SHA-256, SHA-
 730 384 and SHA-512 hash functions.

731 The mechanisms have a parameter, a **CK_RSA_PKCS_PSS_PARAMS** structure. The *sLen* field must
 732 be less than or equal to $k^*-2-hLen$ where *hLen* is the length in bytes of the hash value. k^* is the length in
 733 bytes of the RSA modulus, except if the length in bits of the RSA modulus is one more than a multiple of
 734 8, in which case k^* is one less than the length in bytes of the RSA modulus.

735 Constraints on key types and the length of the data are summarized in the following table. In the table, k
 736 is the length in bytes of the RSA modulus.

737 Table 13, PKCS #1 RSA PSS Signatures with Various Hash Functions: Key And Data Length

Function	Key type	Input length	Output length
C_Sign	RSA private key	any	k
C_Verify	RSA public key	any, k^2	N/A

738 2 Data length, signature length.

739 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
740 specify the supported range of RSA modulus sizes, in bits.

741 2.1.18 PKCS #1 v1.5 RSA signature with SHA3

742 The PKCS #1 v1.5 RSA signature with SHA3-224, SHA3-256, SHA3-384, SHA3-512 mechanisms,
743 denoted **CKM_SHA3_224_RSA_PKCS**, **CKM_SHA3_256_RSA_PKCS**, **CKM_SHA3_384_RSA_PKCS**,
744 and **CKM_SHA3_512_RSA_PKCS** respectively, performs similarly as the other
745 **CKM_SHAX_RSA_PKCS** mechanisms but uses the corresponding SHA3 hash functions.

746 2.1.19 PKCS #1 RSA PSS signature with SHA3

747 The PKCS #1 RSA PSS signature with SHA3-224, SHA3-256, SHA3-384, SHA3-512 mechanisms,
748 denoted **CKM_SHA3_224_RSA_PKCS_PSS**, **CKM_SHA3_256_RSA_PKCS_PSS**,
749 **CKM_SHA3_384_RSA_PKCS_PSS**, and **CKM_SHA3_512_RSA_PKCS_PSS** respectively, performs
750 similarly as the other **CKM_SHAX_RSA_PKCS_PSS** mechanisms but uses the corresponding SHA-3
751 hash functions.

752 2.1.20 ANSI X9.31 RSA signature with SHA-1

753 The ANSI X9.31 RSA signature with SHA-1 mechanism, denoted **CKM_SHA1_RSA_X9_31**, performs
754 single- and multiple-part digital signatures and verification operations without message recovery. The
755 operations performed are as described in ANSI X9.31.

756 This mechanism does not have a parameter.

757 Constraints on key types and the length of the data for these mechanisms are summarized in the
758 following table. In the table, k is the length in bytes of the RSA modulus. For all operations, the k value
759 must be at least 128 and a multiple of 32 as specified in ANSI X9.31.

760 Table 14, ANSI X9.31 RSA Signatures with SHA-1: Key And Data Length

Function	Key type	Input length	Output length
C_Sign	RSA private key	any	k
C_Verify	RSA public key	any, k^2	N/A

761 2 Data length, signature length.

762 For these mechanisms, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO**
763 structure specify the supported range of RSA modulus sizes, in bits.

764 2.1.21 TPM 1.1b and TPM 1.2 PKCS #1 v1.5 RSA

765 The TPM 1.1b and TPM 1.2 PKCS #1 v1.5 RSA mechanism, denoted **CKM_RSA_PKCS TPM_1_1**, is a
766 multi-use mechanism based on the RSA public-key cryptosystem and the block formats initially defined in
767 PKCS #1 v1.5, with additional formatting rules defined in TCPA TPM Specification Version 1.1b.
768 Additional formatting rules remained the same in TCG TPM Specification 1.2. The mechanism supports
769 single-part encryption and decryption; key wrapping; and key unwrapping.

770 This mechanism does not have a parameter. It differs from the standard PKCS#1 v1.5 RSA encryption
771 mechanism in that the plaintext is wrapped in a TCPA_BOUND_DATA (TPM_BOUND_DATA for TPM
772 1.2) structure before being submitted to the PKCS#1 v1.5 encryption process. On encryption, the version
773 field of the TCPA_BOUND_DATA (TPM_BOUND_DATA for TPM 1.2) structure must contain 0x01, 0x01,
774 0x00, 0x00. On decryption, any structure of the form 0x01, 0x01, 0xXX, 0YY may be accepted.

775 This mechanism can wrap and unwrap any secret key of appropriate length. Of course, a particular token
776 may not be able to wrap/unwrap every appropriate-length secret key that it supports. For wrapping, the
777 “input” to the encryption operation is the value of the **CKA_VALUE** attribute of the key that is wrapped;
778 similarly for unwrapping. The mechanism does not wrap the key type or any other information about the
779 key, except the key length; the application must convey these separately. In particular, the mechanism
780 contributes only the **CKA_CLASS** and **CKA_VALUE** (and **CKA_VALUE_LEN**, if the key has it) attributes
781 to the recovered key during unwrapping; other attributes must be specified in the template.

782 Constraints on key types and the length of the data are summarized in the following table. For encryption
783 and decryption, the input and output data may begin at the same location in memory. In the table, k is the
784 length in bytes of the RSA modulus.

785 *Table 15, TPM 1.1b and TPM 1.2 PKCS #1 v1.5 RSA: Key And Data Length*

Function	Key type	Input length	Output length
C_Encrypt ¹	RSA public key	$\leq k-11-5$	k
C_Decrypt ¹	RSA private key	k	$\leq k-11-5$
C_WrapKey	RSA public key	$\leq k-11-5$	k
C_UnwrapKey	RSA private key	k	$\leq k-11-5$

786 ¹ Single-part operations only.

787
788 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
789 specify the supported range of RSA modulus sizes, in bits.

790 2.1.22 TPM 1.1b and TPM 1.2 PKCS #1 RSA OAEP

791 The TPM 1.1b and TPM 1.2 PKCS #1 RSA OAEP mechanism, denoted
792 **CKM_RSA_PKCS_OAEP TPM_1_1**, is a multi-purpose mechanism based on the RSA public-key
793 cryptosystem and the OAEP block format defined in PKCS #1, with additional formatting defined in TCPA
794 TPM Specification Version 1.1b. Additional formatting rules remained the same in TCG TPM
795 Specification 1.2. The mechanism supports single-part encryption and decryption; key wrapping; and key
796 unwrapping.

797 This mechanism does not have a parameter. It differs from the standard PKCS#1 OAEP RSA encryption
798 mechanism in that the plaintext is wrapped in a **TCPA_BOUND_DATA** (**TPM_BOUND_DATA** for TPM
799 1.2) structure before being submitted to the encryption process and that all of the values of the
800 parameters that are passed to a standard **CKM_RSA_PKCS_OAEP** operation are fixed. On encryption,
801 the version field of the **TCPA_BOUND_DATA** (**TPM_BOUND_DATA** for TPM 1.2) structure must contain
802 0x01, 0x01, 0x00, 0x00. On decryption, any structure of the form 0x01, 0x01, 0xXX, 0xYY may be
803 accepted.

804 This mechanism can wrap and unwrap any secret key of appropriate length. Of course, a particular token
805 may not be able to wrap/unwrap every appropriate-length secret key that it supports. For wrapping, the
806 “input” to the encryption operation is the value of the **CKA_VALUE** attribute of the key that is wrapped;
807 similarly for unwrapping. The mechanism does not wrap the key type or any other information about the
808 key, except the key length; the application must convey these separately. In particular, the mechanism
809 contributes only the **CKA_CLASS** and **CKA_VALUE** (and **CKA_VALUE_LEN**, if the key has it) attributes
810 to the recovered key during unwrapping; other attributes must be specified in the template.

811 Constraints on key types and the length of the data are summarized in the following table. For encryption
812 and decryption, the input and output data may begin at the same location in memory. In the table, k is the
813 length in bytes of the RSA modulus.

814 Table 16, TPM 1.1b and TPM 1.2 PKCS #1 RSA OAEP: Key And Data Length

Function	Key type	Input length	Output length
C_Encrypt ¹	RSA public key	$\leq k-2-40-5$	k
C_Decrypt ¹	RSA private key	k	$\leq k-2-40-5$
C_WrapKey	RSA public key	$\leq k-2-40-5$	k
C_UnwrapKey	RSA private key	k	$\leq k-2-40-5$

815 ¹ Single-part operations only.816 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
817 specify the supported range of RSA modulus sizes, in bits.818

2.1.23 RSA AES KEY WRAP

819 The RSA AES key wrap mechanism, denoted **CKM_RSA_AES_KEY_WRAP**, is a mechanism based on
820 the RSA public-key cryptosystem and the AES key wrap mechanism. It supports single-part key
821 wrapping; and key unwrapping.822 It has a parameter, a **CK_RSA_AES_KEY_WRAP_PARAMS** structure.823 The mechanism can wrap and unwrap a target asymmetric key of any length and type using an RSA
824 key.

- 825 - A temporary AES key is used for wrapping the target key using
826 **CKM_AES_KEY_WRAP_KWP** mechanism.
- 827 - The temporary AES key is wrapped with the wrapping RSA key using
828 **CKM_RSA_PKCS_OAEP** mechanism.

830 For wrapping, the mechanism -

- 831 • Generates a temporary random AES key of *ulAESKeyBits* length. This key is not accessible to
832 the user - no handle is returned.
- 833 • Wraps the AES key with the wrapping RSA key using **CKM_RSA_PKCS_OAEP** with parameters
834 of *OAEPPParams*.
- 835 • Wraps the target key with the temporary AES key using **CKM_AES_KEY_WRAP_KWP** ([AES
836 KEYWRAP] section 6.3).
- 837 • Zeroizes the temporary AES key
- 838 • Concatenates two wrapped keys and outputs the concatenated blob. The first is the wrapped
839 AES key, and the second is the wrapped target key.

840 The recommended format for an asymmetric target key being wrapped is as a PKCS8
841 PrivateKeyInfo842 The use of Attributes in the PrivateKeyInfo structure is OPTIONAL. In case of conflicts between the
843 object attribute template, and Attributes in the PrivateKeyInfo structure, an error should be thrown

847 For unwrapping, the mechanism -

- 848 • Splits the input into two parts. The first is the wrapped AES key, and the second is the wrapped
849 target key. The length of the first part is equal to the length of the unwrapping RSA key.
- 850 • Un-wraps the temporary AES key from the first part with the private RSA key using
851 **CKM_RSA_PKCS_OAEP** with parameters of *OAEPPParams*.
- 852 • Un-wraps the target key from the second part with the temporary AES key using
853 **CKM_AES_KEY_WRAP_KWP** ([AES KEYWRAP] section 6.3).

- 854 • Zeroizes the temporary AES key.
 855 • Returns the handle to the newly unwrapped target key.

856 *Table 17, CKM_RSA_AES_KEY_WRAP Mechanisms vs. Functions*

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_RSA_AES_KEY_WRAP						✓	

¹SR = SignRecover, VR = VerifyRecover

2.1.24 RSA AES KEY WRAP mechanism parameters

- ♦ **CK_RSA_AES_KEY_WRAP_PARAMS; CK_RSA_AES_KEY_WRAP_PARAMS_PTR**

CK_RSA_AES_KEY_WRAP_PARAMS is a structure that provides the parameters to the **CKM_RSA_AES_KEY_WRAP** mechanism. It is defined as follows:

```
861     typedef struct CK_RSA_AES_KEY_WRAP_PARAMS {
862         CK ULONG                         ulAESKeyBits;
863         CK_RSA_PKCS_OAEP_PARAMS_PTR      pOAEPParams;
864     } CK_RSA_AES_KEY_WRAP_PARAMS;
```

The fields of the structure have the following meanings:

ulAESKeyBits *length of the temporary AES key in bits. Can be only 128, 192 or 256.*

pOAEPParams *pointer to the parameters of the temporary AES key wrapping. See also the description of PKCS #1 RSA OAEP mechanism parameters.*

CK_RSA_AES_KEY_WRAP_PARAMS_PTR is a pointer to a **CK_RSA_AES_KEY_WRAP_PARAMS**.

2.1.25 FIPS 186-4

When CKM_RSA_PKCS is operated in FIPS mode, the length of the modulus SHALL only be 1024, 2048, or 3072 bits.

2.2 DSA

877 *Table 18, DSA Mechanisms vs. Functions*

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_DSA_KEY_PAIR_GEN					✓		
CKM_DSA_PARAMETER_GEN					✓		

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_DSA_PROBABILISTIC_PARAMETER_GEN					✓		
CKM_DSA_SHAWE_TAYLOR_PARAMETER_GEN					✓		
CKM_DSA_FIPS_G_GEN					✓		
CKM_DSA		✓ ²					
CKM_DSA_SHA1		✓					
CKM_DSA_SHA224		✓					
CKM_DSA_SHA256		✓					
CKM_DSA_SHA384		✓					
CKM_DSA_SHA512		✓					
CKM_DSA_SHA3_224		✓					
CKM_DSA_SHA3_256		✓					
CKM_DSA_SHA3_384		✓					
CKM_DSA_SHA3_512		✓					

878 **2.2.1 Definitions**

879 This section defines the key type "CKK_DSA" for type CK_KEY_TYPE as used in the CKA_KEY_TYPE
 880 attribute of DSA key objects.

881 Mechanisms:

882 CKM_DSA_KEY_PAIR_GEN
 883 CKM_DSA
 884 CKM_DSA_SHA1
 885 CKM_DSA_SHA224
 886 CKM_DSA_SHA256
 887 CKM_DSA_SHA384
 888 CKM_DSA_SHA512
 889 CKM_DSA_SHA3_224
 890 CKM_DSA_SHA3_256
 891 CKM_DSA_SHA3_384
 892 CKM_DSA_SHA3_512
 893 CKM_DSA_PARAMETER_GEN
 894 CKM_DSA_PROBABILISTIC_PARAMETER_GEN
 895 CKM_DSA_SHAWE_TAYLOR_PARAMETER_GEN
 896 CKM_DSA_FIPS_G_GEN

897

898 ◆ **CK_DSA_PARAMETER_GEN_PARAM**

899 CK_DSA_PARAMETER_GEN_PARAM is a structure which provides and returns parameters for the
900 NIST FIPS 186-4 parameter generating algorithms.

901 CK_DSA_PARAMETER_GEN_PARAM_PTR is a pointer to a CK_DSA_PARAMETER_GEN_PARAM.

902

```
903     typedef struct CK_DSA_PARAMETER_GEN_PARAM {  
904         CK_MECHANISM_TYPE    hash;  
905         CK_BYTE_PTR          pSeed;  
906         CK ULONG              ulSeedLen;  
907         CK ULONG              ulIndex;  
908     } CK_DSA_PARAMETER_GEN_PARAM;
```

909 The fields of the structure have the following meanings:

911 *hash* *Mechanism value for the base hash used in PQG generation, Valid
912 values are CKM_SHA_1, CKM_SHA224, CKM_SHA256,
913 CKM_SHA384, CKM_SHA512.*

914 *pSeed* *Seed value used to generate PQ and G. This value is returned by
915 CKM_DSA_PROBABILISTIC_PARAMETER_GEN,
916 CKM_DSA_SHAWE_TAYLOR_PARAMETER_GEN, and passed
917 into CKM_DSA_FIPS_G_GEN.*

918 *ulSeedLen* *Length of seed value.*

919 *ulIndex* *Index value for generating G. Input for CKM_DSA_FIPS_G_GEN.
920 Ignored by CKM_DSA_PROBABILISTIC_PARAMETER_GEN and
921 CKM_DSA_SHAWE_TAYLOR_PARAMETER_GEN.*

922 **2.2.2 DSA public key objects**

923 DSA public key objects (object class **CKO_PUBLIC_KEY**, key type **CKK_DSA**) hold DSA public keys.
924 The following table defines the DSA public key object attributes, in addition to the common attributes
925 defined for this object class:

926 *Table 19, DSA Public Key Object Attributes*

Attribute	Data type	Meaning
CKA_PRIME ^{1,3}	Big integer	Prime p (512 to 3072 bits, in steps of 64 bits)
CKA_SUBPRIME ^{1,3}	Big integer	Subprime q (160, 224 bits, or 256 bits)
CKA_BASE ^{1,3}	Big integer	Base g
CKA_VALUE ^{1,4}	Big integer	Public value y

927 - Refer to [PKCS11-Base] table 11 for footnotes

928 The **CKA_PRIME**, **CKA_SUBPRIME** and **CKA_BASE** attribute values are collectively the “DSA domain
929 parameters”. See FIPS PUB 186-4 for more information on DSA keys.

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

```
931     CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;  
932     CK_KEY_TYPE keyType = CKK_DSA;  
933     CK_UTF8CHAR label[] = "A DSA public key object";  
934     CK_BYTE prime[] = {...};  
935     CK_BYTE subprime[] = {...};
```

```

936     CK_BYTE base[] = { ... };
937     CK_BYTE value[] = { ... };
938     CK_BBOOL true = CK_TRUE;
939     CK_ATTRIBUTE template[] = {
940         { CKA_CLASS, &class, sizeof(class) },
941         { CKA_KEY_TYPE, &keyType, sizeof(keyType) },
942         { CKA_TOKEN, &true, sizeof(true) },
943         { CKA_LABEL, label, sizeof(label)-1 },
944         { CKA_PRIME, prime, sizeof(prime) },
945         { CKA_SUBPRIME, subprime, sizeof(subprime) },
946         { CKA_BASE, base, sizeof(base) },
947         { CKA_VALUE, value, sizeof(value) }
948     };
949

```

950 2.2.3 DSA Key Restrictions

951 FIPS PUB 186-4 specifies permitted combinations of prime and sub-prime lengths. They are:

- Prime: 1024 bits, Subprime: 160
- Prime: 2048 bits, Subprime: 224
- Prime: 2048 bits, Subprime: 256
- Prime: 3072 bits, Subprime: 256

956 Earlier versions of FIPS 186 permitted smaller prime lengths, and those are included here for backwards
957 compatibility. An implementation that is compliant to FIPS 186-4 does not permit the use of primes of
958 any length less than 1024 bits.

959 2.2.4 DSA private key objects

960 DSA private key objects (object class **CKO_PRIVATE_KEY**, key type **CKK_DSA**) hold DSA private keys.
961 The following table defines the DSA private key object attributes, in addition to the common attributes
962 defined for this object class:

963 *Table 20, DSA Private Key Object Attributes*

Attribute	Data type	Meaning
CKA_PRIME ^{1,4,6}	Big integer	Prime p (512 to 1024 bits, in steps of 64 bits)
CKA_SUBPRIME ^{1,4,6}	Big integer	Subprime q (160 bits, 224 bits, or 256 bits)
CKA_BASE ^{1,4,6}	Big integer	Base g
CKA_VALUE ^{1,4,6,7}	Big integer	Private value x

964 - Refer to [PKCS11-Base] table 11 for footnotes

965 The **CKA_PRIME**, **CKA_SUBPRIME** and **CKA_BASE** attribute values are collectively the “DSA domain
966 parameters”. See FIPS PUB 186-4 for more information on DSA keys.

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

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

```

971     CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
972     CK_KEY_TYPE keyType = CKK_DSA;
973     CK_UTF8CHAR label[] = "A DSA private key object";
974     CK_BYTE subject[] = { ... };

```

```

975 CK_BYTE id[] = {123};
976 CK_BYTE prime[] = {...};
977 CK_BYTE subprime[] = {...};
978 CK_BYTE base[] = {...};
979 CK_BYTE value[] = {...};
980 CK_BBOOL true = CK_TRUE;
981 CK_ATTRIBUTE template[] = {
982     {CKA_CLASS, &class, sizeof(class)},
983     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
984     {CKA_TOKEN, &true, sizeof(true)},
985     {CKA_LABEL, label, sizeof(label)-1},
986     {CKA SUBJECT, subject, sizeof(subject)},
987     {CKA_ID, id, sizeof(id)},
988     {CKA_SENSITIVE, &true, sizeof(true)},
989     {CKA_SIGN, &true, sizeof(true)},
990     {CKA_PRIME, prime, sizeof(prime)},
991     {CKA_SUBPRIME, subprime, sizeof(subprime)},
992     {CKA_BASE, base, sizeof(base)},
993     {CKA_VALUE, value, sizeof(value)}}
994 };

```

2.2.5 DSA domain parameter objects

DSA domain parameter objects (object class **CKO_DOMAIN_PARAMETERS**, key type **CKK_DSA**) hold DSA domain parameters. The following table defines the DSA domain parameter object attributes, in addition to the common attributes defined for this object class:

Table 21, DSA Domain Parameter Object Attributes

Attribute	Data type	Meaning
CKA_PRIME ^{1,4}	Big integer	Prime p (512 to 1024 bits, in steps of 64 bits)
CKA_SUBPRIME ^{1,4}	Big integer	Subprime q (160 bits, 224 bits, or 256 bits)
CKA_BASE ^{1,4}	Big integer	Base g
CKA_PRIME_BITS ^{2,3}	CK ULONG	Length of the prime value.

1000 - Refer to [PKCS11-Base] table 11 for footnotes

1001 The **CKA_PRIME**, **CKA_SUBPRIME** and **CKA_BASE** attribute values are collectively the “DSA domain
1002 parameters”. See FIPS PUB 186-4 for more information on DSA domain parameters.

1003 To ensure backwards compatibility, if **CKA_SUBPRIME_BITS** is not specified for a call to
1004 **C_GenerateKey**, it takes on a default based on the value of **CKA_PRIME_BITS** as follows:

- If **CKA_PRIME_BITS** is less than or equal to 1024 then **CKA_SUBPRIME_BITS** shall be 160 bits
- If **CKA_PRIME_BITS** equals 2048 then **CKA_SUBPRIME_BITS** shall be 224 bits
- If **CKA_PRIME_BITS** equals 3072 then **CKA_SUBPRIME_BITS** shall be 256 bits

1009 The following is a sample template for creating a DSA domain parameter object:

```

1010 CK_OBJECT_CLASS class = CKO_DOMAIN_PARAMETERS;
1011 CK_KEY_TYPE keyType = CKK_DSA;
1012 CK_UTF8CHAR label[] = "A DSA domain parameter object";
1013 CK_BYTE prime[] = {...};
1014 CK_BYTE subprime[] = {...};

```

```
1015     CK_BYTE base[] = { ... };
1016     CK_BBOOL true = CK_TRUE;
1017     CK_ATTRIBUTE template[] = {
1018         { CKA_CLASS, &class, sizeof(class) },
1019         { CKA_KEY_TYPE, &keyType, sizeof(keyType) },
1020         { CKA_TOKEN, &true, sizeof(true) },
1021         { CKA_LABEL, label, sizeof(label)-1 },
1022         { CKA_PRIME, prime, sizeof(prime) },
1023         { CKA_SUBPRIME, subprime, sizeof(subprime) },
1024         { CKA_BASE, base, sizeof(base) },
1025     };
```

1026 2.2.6 DSA key pair generation

1027 The DSA key pair generation mechanism, denoted **CKM_DSA_KEY_PAIR_GEN**, is a key pair generation
1028 mechanism based on the Digital Signature Algorithm defined in FIPS PUB 186-2.

1029 This mechanism does not have a parameter.

1030 The mechanism generates DSA public/private key pairs with a particular prime, subprime and base, as
1031 specified in the **CKA_PRIME**, **CKA_SUBPRIME**, and **CKA_BASE** attributes of the template for the public
1032 key.

1033 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new
1034 public key and the **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**, **CKA_SUBPRIME**, **CKA_BASE**, and
1035 **CKA_VALUE** attributes to the new private key. Other attributes supported by the DSA public and private
1036 key types (specifically, the flags indicating which functions the keys support) may also be specified in the
1037 templates for the keys, or else are assigned default initial values.

1038 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1039 specify the supported range of DSA prime sizes, in bits.

1040 2.2.7 DSA domain parameter generation

1041 The DSA domain parameter generation mechanism, denoted **CKM_DSA_PARAMETER_GEN**, is a
1042 domain parameter generation mechanism based on the Digital Signature Algorithm defined in FIPS PUB
1043 186-2.

1044 This mechanism does not have a parameter.

1045 The mechanism generates DSA domain parameters with a particular prime length in bits, as specified in
1046 the **CKA_PRIME_BITS** attribute of the template.

1047 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**, **CKA_SUBPRIME**,
1048 **CKA_BASE** and **CKA_PRIME_BITS** attributes to the new object. Other attributes supported by the DSA
1049 domain parameter types may also be specified in the template, or else are assigned default initial values.

1050 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1051 specify the supported range of DSA prime sizes, in bits.

1052 2.2.8 DSA probabilistic domain parameter generation

1053 The DSA probabilistic domain parameter generation mechanism, denoted
1054 **CKM_DSA_PROBABILISTIC_PARAMETER_GEN**, is a domain parameter generation mechanism based
1055 on the Digital Signature Algorithm defined in FIPS PUB 186-4, section Appendix A.1.1 Generation and
1056 Validation of Probable Primes..

1057 This mechanism takes a **CK_DSA_PARAMETER_GEN_PARAM** which supplies the base hash and
1058 returns the seed (pSeed) and the length (ulSeedLen).

1059 The mechanism generates DSA the prime and subprime domain parameters with a particular prime
1060 length in bits, as specified in the **CKA_PRIME_BITS** attribute of the template and the subprime length as
1061 specified in the **CKA_SUBPRIME_BITS** attribute of the template.
1062 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**, **CKA_SUBPRIME**,
1063 **CKA_PRIME_BITS**, and **CKA_SUBPRIME_BITS** attributes to the new object. **CKA_BASE** is not set by
1064 this call. Other attributes supported by the DSA domain parameter types may also be specified in the
1065 template, or else are assigned default initial values.
1066 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1067 specify the supported range of DSA prime sizes, in bits.

1068 **2.2.9 DSA Shawe-Taylor domain parameter generation**

1069 The DSA Shawe-Taylor domain parameter generation mechanism, denoted
1070 **CKM_DSA_SHAWE_TAYLOR_PARAMETER_GEN**, is a domain parameter generation mechanism
1071 based on the Digital Signature Algorithm defined in FIPS PUB 186-4, section Appendix A.1.2
1072 Construction and Validation of Provable Primes p and q.
1073 This mechanism takes a **CK_DSA_PARAMETER_GEN_PARAM** which supplies the base hash and
1074 returns the seed (pSeed) and the length (ulSeedLen).
1075 The mechanism generates DSA the prime and subprime domain parameters with a particular prime
1076 length in bits, as specified in the **CKA_PRIME_BITS** attribute of the template and the subprime length as
1077 specified in the **CKA_SUBPRIME_BITS** attribute of the template.
1078 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**, **CKA_SUBPRIME**,
1079 **CKA_PRIME_BITS**, and **CKA_SUBPRIME_BITS** attributes to the new object. **CKA_BASE** is not set by
1080 this call. Other attributes supported by the DSA domain parameter types may also be specified in the
1081 template, or else are assigned default initial values.
1082 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1083 specify the supported range of DSA prime sizes, in bits.

1084 **2.2.10 DSA base domain parameter generation**

1085 The DSA base domain parameter generation mechanism, denoted **CKM_DSA_FIPS_G_GEN**, is a base
1086 parameter generation mechanism based on the Digital Signature Algorithm defined in FIPS PUB 186-4,
1087 section Appendix A.2 Generation of Generator G.
1088 This mechanism takes a **CK_DSA_PARAMETER_GEN_PARAM** which supplies the base hash the seed
1089 (pSeed) and the length (ulSeedLen) and the index value.
1090 The mechanism generates the DSA base with the domain parameter specified in the **CKA_PRIME** and
1091 **CKA_SUBPRIME** attributes of the template.
1092 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_BASE** attributes to the new
1093 object. Other attributes supported by the DSA domain parameter types may also be specified in the
1094 template, or else are assigned default initial values.
1095 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1096 specify the supported range of DSA prime sizes, in bits.

1097 **2.2.11 DSA without hashing**

1098 The DSA without hashing mechanism, denoted **CKM_DSA**, is a mechanism for single-part signatures and
1099 verification based on the Digital Signature Algorithm defined in FIPS PUB 186-2. (This mechanism
1100 corresponds only to the part of DSA that processes the 20-byte hash value; it does not compute the hash
1101 value.)
1102 For the purposes of this mechanism, a DSA signature is a 40-byte string, corresponding to the
1103 concatenation of the DSA values *r* and *s*, each represented most-significant byte first.
1104 It does not have a parameter.
1105 Constraints on key types and the length of data are summarized in the following table:

1106 *Table 22, DSA: Key And Data Length*

Function	Key type	Input length	Output length
C_Sign ¹	DSA private key	20, 28, 32, 48, or 64 bits	2*length of subprime
C_Verify ¹	DSA public key	(20, 28, 32, 48, or 64 bits), (2*length of subprime) ²	N/A

1107 ¹ Single-part operations only.1108 ² Data length, signature length.1109 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1110 specify the supported range of DSA prime sizes, in bits.

1111 2.2.12 DSA with SHA-1

1112 The DSA with SHA-1 mechanism, denoted **CKM_DSA_SHA1**, is a mechanism for single- and multiple-
1113 part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB 186-2.
1114 This mechanism computes the entire DSA specification, including the hashing with SHA-1.1115 For the purposes of this mechanism, a DSA signature is a 40-byte string, corresponding to the
1116 concatenation of the DSA values *r* and *s*, each represented most-significant byte first.

1117 This mechanism does not have a parameter.

1118 Constraints on key types and the length of data are summarized in the following table:

1119 *Table 23, DSA with SHA-1: Key And Data Length*

Function	Key type	Input length	Output length
C_Sign	DSA private key	any	2*subprime length
C_Verify	DSA public key	any, 2*subprime length ²	N/A

1120 ² Data length, signature length.1121 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1122 specify the supported range of DSA prime sizes, in bits.

1123 2.2.13 FIPS 186-4

1124 When CKM_DSA is operated in FIPS mode, only the following bit lengths of *p* and *q*, represented by *L*
1125 and *N*, SHALL be used:1126 *L* = 1024, *N* = 1601127 *L* = 2048, *N* = 2241128 *L* = 2048, *N* = 2561129 *L* = 3072, *N* = 256

1130

1131 2.2.14 DSA with SHA-224

1132 The DSA with SHA-1 mechanism, denoted **CKM_DSA_SHA224**, is a mechanism for single- and multiple-
1133 part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB 186-4.
1134 This mechanism computes the entire DSA specification, including the hashing with SHA-224.

1135 For the purposes of this mechanism, a DSA signature is a string of length 2^{*}subprime , corresponding to
 1136 the concatenation of the DSA values r and s , each represented most-significant byte first.
 1137 This mechanism does not have a parameter.
 1138 Constraints on key types and the length of data are summarized in the following table:
 1139 *Table 24, DSA with SHA-244: Key And Data Length*

Function	Key type	Input length	Output length
C_Sign	DSA private key	any	2^{*}subprime length
C_Verify	DSA public key	any, 2^{*}subprime length ²	N/A

1140 ² Data length, signature length.
 1141 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 1142 specify the supported range of DSA prime sizes, in bits.

2.2.15 DSA with SHA-256

1144 The DSA with SHA-1 mechanism, denoted **CKM_DSA_SHA256**, is a mechanism for single- and multiple-
 1145 part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB 186-4.
 1146 This mechanism computes the entire DSA specification, including the hashing with SHA-256.
 1147 For the purposes of this mechanism, a DSA signature is a string of length 2^{*}subprime , corresponding to
 1148 the concatenation of the DSA values r and s , each represented most-significant byte first.
 1149 This mechanism does not have a parameter.
 1150 Constraints on key types and the length of data are summarized in the following table:
 1151 *Table 25, DSA with SHA-256: Key And Data Length*

Function	Key type	Input length	Output length
C_Sign	DSA private key	any	2^{*}subprime length
C_Verify	DSA public key	any, 2^{*}subprime length ²	N/A

1152 ² Data length, signature length.
 1153

2.2.16 DSA with SHA-384

 1154 The DSA with SHA-1 mechanism, denoted **CKM_DSA_SHA384**, is a mechanism for single- and multiple-
 1155 part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB 186-4.
 1156 This mechanism computes the entire DSA specification, including the hashing with SHA-384.
 1157 For the purposes of this mechanism, a DSA signature is a string of length 2^{*}subprime , corresponding to
 1158 the concatenation of the DSA values r and s , each represented most-significant byte first.
 1159 This mechanism does not have a parameter.
 1160 Constraints on key types and the length of data are summarized in the following table:

1161 Table 26, DSA with SHA-384: Key And Data Length

Function	Key type	Input length	Output length
C_Sign	DSA private key	any	2*subprime length
C_Verify	DSA public key	any, 2*subprime length ²	N/A

1162 ² Data length, signature length.

2.2.17 DSA with SHA-512

1164 The DSA with SHA-1 mechanism, denoted **CKM_DSA_SHA512**, is a mechanism for single- and multiple-part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB 186-4.
1165 This mechanism computes the entire DSA specification, including the hashing with SHA-512.

1167 For the purposes of this mechanism, a DSA signature is a string of length 2*subprime, corresponding to
1168 the concatenation of the DSA values *r* and *s*, each represented most-significant byte first.

1169 This mechanism does not have a parameter.

1170 Constraints on key types and the length of data are summarized in the following table:

1171 Table 27, DSA with SHA-512: Key And Data Length

Function	Key type	Input length	Output length
C_Sign	DSA private key	any	2*subprime length
C_Verify	DSA public key	any, 2*subprime length ²	N/A

1172 ² Data length, signature length.

2.2.18 DSA with SHA3-224

1174 The DSA with SHA3-224 mechanism, denoted **CKM_DSA_SHA3_224**, is a mechanism for single- and
1175 multiple-part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB
1176 186-4. This mechanism computes the entire DSA specification, including the hashing with SHA3-224.

1177 For the purposes of this mechanism, a DSA signature is a string of length 2*subprime, corresponding to
1178 the concatenation of the DSA values *r* and *s*, each represented most-significant byte first.

1179 This mechanism does not have a parameter.

1180 Constraints on key types and the length of data are summarized in the following table:

1181 Table 28, DSA with SHA3-224: Key And Data Length

Function	Key type	Input length	Output length
C_Sign	DSA private key	any	2*subprime length
C_Verify	DSA public key	any, 2*subprime length ²	N/A

1182 ² Data length, signature length.

1183 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1184 specify the supported range of DSA prime sizes, in bits.

1185 2.2.19 DSA with SHA3-256

1186 The DSA with SHA3-256 mechanism, denoted **CKM_DSA_SHA3_256**, is a mechanism for single- and
1187 multiple-part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB
1188 186-4. This mechanism computes the entire DSA specification, including the hashing with SHA3-256.

1189 For the purposes of this mechanism, a DSA signature is a string of length 2^{*} subprime, corresponding to
1190 the concatenation of the DSA values r and s , each represented most-significant byte first.

1191 This mechanism does not have a parameter.

1192 Constraints on key types and the length of data are summarized in the following table:

1193 *Table 29, DSA with SHA3-256: Key And Data Length*

Function	Key type	Input length	Output length
C_Sign	DSA private key	any	2^{*} subprime length
C_Verify	DSA public key	any, 2^{*} subprime length ²	N/A

1194 ² Data length, signature length.

1195 2.2.20 DSA with SHA3-384

1196 The DSA with SHA3-384 mechanism, denoted **CKM_DSA_SHA3_384**, is a mechanism for single- and
1197 multiple-part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB
1198 186-4. This mechanism computes the entire DSA specification, including the hashing with SHA3-384.

1199 For the purposes of this mechanism, a DSA signature is a string of length 2^{*} subprime, corresponding to
1200 the concatenation of the DSA values r and s , each represented most-significant byte first.

1201 This mechanism does not have a parameter.

1202 Constraints on key types and the length of data are summarized in the following table:

1203 *Table 30, DSA with SHA3-384: Key And Data Length*

Function	Key type	Input length	Output length
C_Sign	DSA private key	any	2^{*} subprime length
C_Verify	DSA public key	any, 2^{*} subprime length ²	N/A

1204 ² Data length, signature length.

1205 2.2.21 DSA with SHA3-512

1206 The DSA with SHA3-512 mechanism, denoted **CKM_DSA_SHA3_512**, is a mechanism for single- and
1207 multiple-part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB
1208 186-4. This mechanism computes the entire DSA specification, including the hashing with SH3A-512.

1209 For the purposes of this mechanism, a DSA signature is a string of length 2^{*} subprime, corresponding to
1210 the concatenation of the DSA values r and s , each represented most-significant byte first.

1211 This mechanism does not have a parameter.

1212 Constraints on key types and the length of data are summarized in the following table:

1213 Table 31, DSA with SHA3-512: Key And Data Length

Function	Key type	Input length	Output length
C_Sign	DSA private key	any	2*subprime length
C_Verify	DSA public key	any, 2*subprime length ²	N/A

1214 ² Data length, signature length.

1215

1216

2.3 Elliptic Curve

1217 The Elliptic Curve (EC) cryptosystem (also related to ECDSA) in this document was originally based on
1218 the one described in the ANSI X9.62 and X9.63 standards developed by the ANSI X9F1 working group.1219 The EC cryptosystem developed by the ANSI X9F1 working group was created at a time when EC curves
1220 were always represented in their Weierstrass form. Since that time, new curves represented in Edwards
1221 form (RFC 8032) and Montgomery form (RFC 7748) have become more common. To support these new
1222 curves, the EC cryptosystem in this document has been extended from the original. Additional key
1223 generation mechanisms have been added as well as an additional signature generation mechanism.

1224

1225 Table 32, Elliptic Curve Mechanisms vs. Functions

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_EC_KEY_PAIR_GEN					✓		
CKM_EC_KEY_PAIR_GEN_W_EXTRA_BITS					✓		
CKM_EC_EDWARDS_KEY_PAIR_GEN					✓		
CKM_EC_MONTGOMERY_KEY_PAIR_GEN					✓		
CKM_ECDSA		✓ ²					
CKM_ECDSA_SHA1		✓					
CKM_ECDSA_SHA224		✓					
CKM_ECDSA_SHA256		✓					
CKM_ECDSA_SHA384		✓					
CKM_ECDSA_SHA512		✓					
CKM_ECDSA_SHA3_224		✓					
CKM_ECDSA_SHA3_256		✓					
CKM_ECDSA_SHA3_384		✓					
CKM_ECDSA_SHA3_512		✓					
CKM_EDDSA		✓					
CKM_XEDDSA		✓					
CKM_ECDH1_DERIVE							✓
CKM_ECDH1_COFACTOR_DERIVE							✓

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR ¹	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_ECMQV_DERIVE							✓
CKM_ECDH_AES_KEY_WRAP						✓	

1226

1227 *Table 33, Mechanism Information Flags*

CKF_EC_F_P	0x00100000UL	True if the mechanism can be used with EC domain parameters over F_p
CKF_EC_F_2M	0x00200000UL	True if the mechanism can be used with EC domain parameters over F_{2^m}
CKF_EC_ECPARAMETERS	0x00400000UL	True if the mechanism can be used with EC domain parameters of the choice ecParameters
CKF_EC_OID	0x00800000UL	True if the mechanism can be used with EC domain parameters of the choice old
CKF_EC_UNCOMPRESS	0x01000000UL	True if the mechanism can be used with elliptic curve point uncompressed
CKF_EC_COMPRESS	0x02000000UL	True if the mechanism can be used with elliptic curve point compressed
CKF_EC_CURVENAME	0x04000000UL	True of the mechanism can be used with EC domain parameters of the choice curveName

1228 Note: CKF_EC_NAMEDCURVE is deprecated with PKCS#11 3.00. It is replaced by CKF_EC_OID.

1229 In these standards, there are two different varieties of EC defined:

1230 1. EC using a field with an odd prime number of elements (i.e. the finite field F_p).1231 2. EC using a field of characteristic two (i.e. the finite field F_{2^m}).1232 An EC key in Cryptoki contains information about which variety of EC it is suited for. It is preferable that a
1233 Cryptoki library, which can perform EC mechanisms, be capable of performing operations with the two
1234 varieties of EC, however this is not required. The **CK_MECHANISM_INFO** structure **CKF_EC_F_P** flag
1235 identifies a Cryptoki library supporting EC keys over F_p whereas the **CKF_EC_F_2M** flag identifies a
1236 Cryptoki library supporting EC keys over F_{2^m} . A Cryptoki library that can perform EC mechanisms must
1237 set either or both of these flags for each EC mechanism.1238 In these specifications there are also four representation methods to define the domain parameters for an
1239 EC key. Only the **ecParameters**, the **old** and the **curveName** choices are supported in Cryptoki. The
1240 **CK_MECHANISM_INFO** structure **CKF_EC_ECPARAMETERS** flag identifies a Cryptoki library
1241 supporting the **ecParameters** choice whereas the **CKF_EC_OID** flag identifies a Cryptoki library
1242 supporting the **old** choice, and the **CKF_EC_CURVENAME** flag identifies a Cryptoki library supporting
1243 the **curveName** choice. A Cryptoki library that can perform EC mechanisms must set the appropriate
1244 flag(s) for each EC mechanism.1245 In these specifications, an EC public key (i.e. EC point Q) or the base point G when the **ecParameters**
1246 choice is used can be represented as an octet string of the uncompressed form or the compressed form.
1247 The **CK_MECHANISM_INFO** structure **CKF_EC_UNCOMPRESS** flag identifies a Cryptoki library
1248 supporting the uncompressed form whereas the **CKF_EC_COMPRESS** flag identifies a Cryptoki library
1249 supporting the compressed form. A Cryptoki library that can perform EC mechanisms must set either or
1250 both of these flags for each EC mechanism.

1251 Note that an implementation of a Cryptoki library supporting EC with only one variety, one representation
1252 of domain parameters or one form may encounter difficulties achieving interoperability with other
1253 implementations.
1254 If an attempt to create, generate, derive or unwrap an EC key of an unsupported curve is made, the
1255 attempt should fail with the error code CKR_CURVE_NOT_SUPPORTED. If an attempt to create,
1256 generate, derive, or unwrap an EC key with invalid or of an unsupported representation of domain
1257 parameters is made, that attempt should fail with the error code CKR_DOMAIN_PARAMS_INVALID. If
1258 an attempt to create, generate, derive, or unwrap an EC key of an unsupported form is made, that
1259 attempt should fail with the error code CKR_TEMPLATE_INCONSISTENT.

1260 **2.3.1 EC Signatures**

1261 For the purposes of these mechanisms, an ECDSA signature is an octet string of even length which is at
1262 most two times $nLen$ octets, where $nLen$ is the length in octets of the base point order n . The signature
1263 octets correspond to the concatenation of the ECDSA values r and s , both represented as an octet string
1264 of equal length of at most $nLen$ with the most significant byte first. If r and s have different octet length,
1265 the shorter of both must be padded with leading zero octets such that both have the same octet length.
1266 Loosely spoken, the first half of the signature is r and the second half is s . For signatures created by a
1267 token, the resulting signature is always of length $2nLen$. For signatures passed to a token for verification,
1268 the signature may have a shorter length but must be composed as specified before.

1269 If the length of the hash value is larger than the bit length of n , only the leftmost bits of the hash up to the
1270 length of n will be used. Any truncation is done by the token.

1271 Note: For applications, it is recommended to encode the signature as an octet string of length two times
1272 $nLen$ if possible. This ensures that the application works with PKCS#11 modules which have been
1273 implemented based on an older version of this document. Older versions required all signatures to have
1274 length two times $nLen$. It may be impossible to encode the signature with the maximum length of two
1275 times $nLen$ if the application just gets the integer values of r and s (i.e. without leading zeros), but does
1276 not know the base point order n , because r and s can have any value between zero and the base point
1277 order n .

1278 An EdDSA signature is an octet string of even length which is two times $nLen$ octets, where $nLen$ is
1279 calculated as EdDSA parameter b divided by 8. The signature octets correspond to the concatenation of
1280 the EdDSA values R and S as defined in [RFC 8032], both represented as an octet string of equal length
1281 of $nLen$ bytes in little endian order.

1282 **2.3.2 Definitions**

1283 This section defines the key type “CKK_EC” for type CK_KEY_TYPE as used in the CKA_KEY_TYPE
1284 attribute of key objects.

1285 Note: CKK_ECDSA is deprecated. It is replaced by CKK_EC.

1286 Mechanisms:

1287

1288 CKM_EC_KEY_PAIR_GEN
1289 CKM_EC_EDWARDS_KEY_PAIR_GEN
1290 CKM_EC_MONTGOMERY_KEY_PAIR_GEN
1291 CKM_ECDSA
1292 CKM_ECDSA_SHA1
1293 CKM_ECDSA_SHA224
1294 CKM_ECDSA_SHA256
1295 CKM_ECDSA_SHA384
1296 CKM_ECDSA_SHA512
1297 CKM_ECDSA_SHA3_224

1298 CKM_ECDSA_SHA3_256
1299 CKM_ECDSA_SHA3_384
1300 CKM_ECDSA_SHA3_512
1301 CKM_EDDSA
1302 CKM_XEDDSA
1303 CKM_ECDH1_DERIVE
1304 CKM_ECDH1_COFACTOR_DERIVE
1305 CKM_ECMQV_DERIVE
1306 CKM_ECDH_AES_KEY_WRAP
1307
1308 CKD_NULL
1309 CKD_SHA1_KDF
1310 CKD_SHA224_KDF
1311 CKD_SHA256_KDF
1312 CKD_SHA384_KDF
1313 CKD_SHA512_KDF
1314 CKD_SHA3_224_KDF
1315 CKD_SHA3_256_KDF
1316 CKD_SHA3_384_KDF
1317 CKD_SHA3_512_KDF
1318 CKD_SHA1_KDF_SP800
1319 CKD_SHA224_KDF_SP800
1320 CKD_SHA256_KDF_SP800
1321 CKD_SHA384_KDF_SP800
1322 CKD_SHA512_KDF_SP800
1323 CKD_SHA3_224_KDF_SP800
1324 CKD_SHA3_256_KDF_SP800
1325 CKD_SHA3_384_KDF_SP800
1326 CKD_SHA3_512_KDF_SP800
1327 CKD_BLAKE2B_160_KDF
1328 CKD_BLAKE2B_256_KDF
1329 CKD_BLAKE2B_384_KDF
1330 CKD_BLAKE2B_512_KDF

1331 2.3.3 ECDSA public key objects

1332 EC (also related to ECDSA) public key objects (object class **CKO_PUBLIC_KEY**, key type **CKK_EC**)
1333 hold EC public keys. The following table defines the EC public key object attributes, in addition to the
1334 common attributes defined for this object class:

1335 Table 34, Elliptic Curve Public Key Object Attributes

Attribute	Data type	Meaning
CKA_EC_PARAMS ^{1,3}	Byte array	DER-encoding of an ANSI X9.62 Parameters value
CKA_EC_POINT ^{1,4}	Byte array	DER-encoding of ANSI X9.62 ECPoint value Q

1336 - Refer to [PKCS11-Base] table 11 for footnotes

1337 Note: CKA_ECDSA_PARAMS is deprecated. It is replaced by CKA_EC_PARAMS.

1338 The **CKA_EC_PARAMS** attribute value is known as the “EC domain parameters” and is defined in ANSI
1339 X9.62 as a choice of three parameter representation methods with the following syntax:

```

1340     Parameters ::= CHOICE {
1341         ecParameters    ECParameters,
1342         oid            CURVES.&id({CurveNames}),
1343         implicitlyCA   NULL,
1344         curveName      PrintableString
1345     }

```

1346

1347 This allows detailed specification of all required values using choice **ecParameters**, the use of **oid** as an
1348 object identifier substitute for a particular set of elliptic curve domain parameters, or **implicitlyCA** to
1349 indicate that the domain parameters are explicitly defined elsewhere, or **curveName** to specify a curve
1350 name as e.g. define in [ANSI X9.62], [BRAINPOOL], [SEC 2], [LEGIFRANCE]. The use of **oid** or
1351 **curveName** is recommended over the choice **ecParameters**. The choice **implicitlyCA** must not be used
1352 in Cryptoki.

1353 The following is a sample template for creating an EC (ECDSA) public key object:

```

1354     CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
1355     CK_KEY_TYPE keyType = CKK_EC;
1356     CK_UTF8CHAR label[] = "An EC public key object";
1357     CK_BYTE ecParams[] = {...};
1358     CK_BYTE ecPoint[] = {...};
1359     CK_BBOOL true = CK_TRUE;
1360     CK_ATTRIBUTE template[] = {
1361         {CKA_CLASS, &class, sizeof(class)},
1362         {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1363         {CKA_TOKEN, &true, sizeof(true)},
1364         {CKA_LABEL, label, sizeof(label)-1},
1365         {CKA_EC_PARAMS, ecParams, sizeof(ecParams)},
1366         {CKA_EC_POINT, ecPoint, sizeof(ecPoint)}}
1367     };

```

1368

2.3.4 Elliptic curve private key objects

1369 EC (also related to ECDSA) private key objects (object class **CKO_PRIVATE_KEY**, key type **CKK_EC**)
1370 hold EC private keys. See Section 2.3 for more information about EC. The following table defines the EC
1371 private key object attributes, in addition to the common attributes defined for this object class:

1372 Table 35, Elliptic Curve Private Key Object Attributes

Attribute	Data type	Meaning
CKA_EC_PARAMS ^{1,4,6}	Byte array	DER-encoding of an ANSI X9.62 Parameters value
CKA_VALUE ^{1,4,6,7}	Big integer	ANSI X9.62 private value d

1373 - Refer to [PKCS11-Base] table 11 for footnotes

1374 The **CKA_EC_PARAMS** attribute value is known as the “EC domain parameters” and is defined in ANSI
1375 X9.62 as a choice of three parameter representation methods with the following syntax:

```

1376     Parameters ::= CHOICE {
1377         ecParameters    ECParameters,
1378         oid             CURVES.&id({CurveNames}),
1379         implicitlyCA   NULL,
1380         curveName      PrintableString
1381     }
1382

```

1383 This allows detailed specification of all required values using choice **ecParameters**, the use of **oid** as an
1384 object identifier substitute for a particular set of elliptic curve domain parameters, or **implicitlyCA** to
1385 indicate that the domain parameters are explicitly defined elsewhere, or **curveName** to specify a curve
1386 name as e.g. define in [ANSI X9.62], [BRAINPOOL], [SEC 2], [LEGIFRANCE]. The use of **oid** or
1387 **curveName** is recommended over the choice **ecParameters**. The choice **implicitlyCA** must not be used
1388 in Cryptoki. Note that when generating an EC private key, the EC domain parameters are *not* specified in
1389 the key's template. This is because EC private keys are only generated as part of an EC key pair, and
1390 the EC domain parameters for the pair are specified in the template for the EC public key.

1391 The following is a sample template for creating an EC (ECDSA) private key object:

```

1392     CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
1393     CK_KEY_TYPE keyType = CKK_EC;
1394     CK_UTF8CHAR label[] = "An EC private key object";
1395     CK_BYTE subject[] = {...};
1396     CK_BYTE id[] = {123};
1397     CK_BYTE ecParams[] = {...};
1398     CK_BYTE value[] = {...};
1399     CK_BBOOL true = CK_TRUE;
1400     CK_ATTRIBUTE template[] = {
1401         {CKA_CLASS, &class, sizeof(class)},
1402         {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1403         {CKA_TOKEN, &true, sizeof(true)},
1404         {CKA_LABEL, label, sizeof(label)-1},
1405         {CKA_SUBJECT, subject, sizeof(subject)},
1406         {CKA_ID, id, sizeof(id)},
1407         {CKA_SENSITIVE, &true, sizeof(true)},
1408         {CKA_DERIVE, &true, sizeof(true)},
1409         {CKA_EC_PARAMS, ecParams, sizeof(ecParams)},
1410         {CKA_VALUE, value, sizeof(value)}
1411     };

```

2.3.5 Edwards Elliptic curve public key objects

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

Table 36, Edwards Elliptic Curve Public Key Object Attributes

Attribute	Data type	Meaning
CKA_EC_PARAMS ^{1,3}	Byte array	DER-encoding of a Parameters value as defined above
CKA_EC_POINT ^{1,4}	Byte array	DER-encoding of the b-bit public key value in little endian order as defined in RFC 8032

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

The **CKA_EC_PARAMS** attribute value is known as the “EC domain parameters” and is defined in ANSI X9.62 as a choice of three parameter representation methods. A 4th choice is added to support Edwards and Montgomery Elliptic curves. The **CKA_EC_PARAMS** attribute has the following syntax:

```
Parameters ::= CHOICE {
    ecParameters  ECParameters,
    oid           CURVES.&id({CurveNames}),
    implicitlyCA NULL,
    curveName     PrintableString
}
```

Edwards EC public keys only support the use of the **curveName** selection to specify a curve name as defined in [RFC 8032] and the use of the **oid** selection to specify a curve through an EdDSA algorithm as defined in [RFC 8410]. Note that keys defined by RFC 8032 and RFC 8410 are incompatible.

The following is a sample template for creating an Edwards EC public key object with Edwards25519 being specified as **curveName**:

```
CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_EC;
CK_UTF8CHAR label[] = "An Edwards EC public key object";
CK_BYTE ecParams[] = {0x13, 0x0c, 0x65, 0x64, 0x77, 0x61,
                      0x72, 0x64, 0x73, 0x32, 0x35, 0x35, 0x31, 0x39};
CK_BYTE ecPoint[] = {...};
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_EC_PARAMS, ecParams, sizeof(ecParams)},
    {CKA_EC_POINT, ecPoint, sizeof(ecPoint)}};
};
```

2.3.6 Edwards Elliptic curve private key objects

Edwards EC private key objects (object class **CKO_PRIVATE_KEY**, key type **CKK_EC_EDWARDS**) hold Edwards EC private keys. See Section 2.3 for more information about EC. The following table defines the Edwards EC private key object attributes, in addition to the common attributes defined for this object class:

1452 Table 37, Edwards Elliptic Curve Private Key Object Attributes

Attribute	Data type	Meaning
CKA_EC_PARAMS ^{1,4,6}	Byte array	DER-encoding of a Parameters value as defined above
CKA_VALUE ^{1,4,6,7}	Big integer	b-bit private key value in little endian order as defined in RFC 8032

1453 - Refer to [PKCS #11-Base] table 11 for footnotes

1454 The **CKA_EC_PARAMS** attribute value is known as the “EC domain parameters” and is defined in ANSI
1455 X9.62 as a choice of three parameter representation methods. A 4th choice is added to support Edwards
1456 and Montgomery Elliptic curves. The CKA_EC_PARAMS attribute has the following syntax:

```
1457     Parameters ::= CHOICE {  
1458         ecParameters    ECParameters,  
1459         oId             CURVES.&id({CurveNames}),  
1460         implicitlyCA   NULL,  
1461         curveName      PrintableString  
1462     }
```

1463 Edwards EC private keys only support the use of the **curveName** selection to specify a curve name as
1464 defined in [RFC 8032] and the use of the **oId** selection to specify a curve through an EdDSA algorithm as
1465 defined in [RFC 8410]. Note that keys defined by RFC 8032 and RFC 8410 are incompatible.

1466 Note that when generating an Edwards EC private key, the EC domain parameters are *not* specified in
1467 the key's template. This is because Edwards EC private keys are only generated as part of an Edwards
1468 EC key pair, and the EC domain parameters for the pair are specified in the template for the Edwards EC
1469 public key.

1470 The following is a sample template for creating an Edwards EC private key object:

```
1471     CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;  
1472     CK_KEY_TYPE keyType = CKK_EC;  
1473     CK_UTF8CHAR label[] = "An Edwards EC private key object";  
1474     CK_BYTE subject[] = {...};  
1475     CK_BYTE id[] = {123};  
1476     CK_BYTE ecParams[] = {...};  
1477     CK_BYTE value[] = {...};  
1478     CK_BBOOL true = CK_TRUE;  
1479     CK_ATTRIBUTE template[] = {  
1480         {CKA_CLASS, &class, sizeof(class)},  
1481         {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1482         {CKA_TOKEN, &true, sizeof(true)},  
1483         {CKA_LABEL, label, sizeof(label)-1},  
1484         {CKA_SUBJECT, subject, sizeof(subject)},  
1485         {CKA_ID, id, sizeof(id)},  
1486         {CKA_SENSITIVE, &true, sizeof(true)},  
1487         {CKA_DERIVE, &true, sizeof(true)},  
1488         {CKA_VALUE, value, sizeof(value)}  
1489     };
```

1490 2.3.7 Montgomery Elliptic curve public key objects

1491 Montgomery EC public key objects (object class **CKO_PUBLIC_KEY**, key type
1492 **CKK_EC_MONTGOMERY**) hold Montgomery EC public keys. The following table defines the

1493 Montgomery EC public key object attributes, in addition to the common attributes defined for this object
1494 class:

1495 *Table 38, Montgomery Elliptic Curve Public Key Object Attributes*

Attribute	Data type	Meaning
CKA_EC_PARAMS ^{1,3}	Byte array	DER-encoding of a Parameters value as defined above
CKA_EC_POINT ^{1,4}	Byte array	DER-encoding of the public key value in little endian order as defined in RFC 7748

1496 - Refer to [PKCS #11-Base] table 11 for footnotes

1497 The **CKA_EC_PARAMS** attribute value is known as the “EC domain parameters” and is defined in ANSI
1498 X9.62 as a choice of three parameter representation methods. A 4th choice is added to support Edwards
1499 and Montgomery Elliptic curves. The CKA_EC_PARAMS attribute has the following syntax:

```
1500     Parameters ::= CHOICE {  
1501         ecParameters    ECParameters,  
1502         oId              CURVES.&id({CurveNames}),  
1503         implicitlyCA    NULL,  
1504         curveName       PrintableString  
1505     }
```

1506 Montgomery EC public keys only support the use of the **curveName** selection to specify a curve name as
1507 defined in [RFC7748] and the use of the **oId** selection to specify a curve through an ECDH algorithm as
1508 defined in [RFC 8410]. Note that keys defined by RFC 7748 and RFC 8410 are incompatible.

1509 The following is a sample template for creating a Montgomery EC public key object:

```
1510     CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;  
1511     CK_KEY_TYPE keyType = CKK_EC;  
1512     CK_UTF8CHAR label[] = "A Montgomery EC public key object";  
1513     CK_BYTE ecParams[] = {...};  
1514     CK_BYTE ecPoint[] = {...};  
1515     CK_BBOOL true = CK_TRUE;  
1516     CK_ATTRIBUTE template[] = {  
1517         {CKA_CLASS, &class, sizeof(class)},  
1518         {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
1519         {CKA_TOKEN, &true, sizeof(true)},  
1520         {CKA_LABEL, label, sizeof(label)-1},  
1521         {CKA_EC_PARAMS, ecParams, sizeof(ecParams)},  
1522         {CKA_EC_POINT, ecPoint, sizeof(ecPoint)}  
1523     };
```

1524 2.3.8 Montgomery Elliptic curve private key objects

1525 Montgomery EC private key objects (object class **CKO_PRIVATE_KEY**, key type
1526 **CKK_EC_MONTGOMERY**) hold Montgomery EC private keys. See Section 2.3 for more information
1527 about EC. The following table defines the Montgomery EC private key object attributes, in addition to the
1528 common attributes defined for this object class:

1529 Table 39, Montgomery Elliptic Curve Private Key Object Attributes

Attribute	Data type	Meaning
CKA_EC_PARAMS ^{1,4,6}	Byte array	DER-encoding of a Parameters value as defined above
CKA_VALUE ^{1,4,6,7}	Big integer	Private key value in little endian order as defined in RFC 7748

1530 - Refer to [PKCS #11-Base] table 11 for footnotes

1531 The **CKA_EC_PARAMS** attribute value is known as the “EC domain parameters” and is defined in ANSI
 1532 X9.62 as a choice of three parameter representation methods. A 4th choice is added to support Edwards
 1533 and Montgomery Elliptic curves. The CKA_EC_PARAMS attribute has the following syntax:

```
1534     Parameters ::= CHOICE {
1535         ecParameters    ECParameters,
1536         oId             CURVES.&id({CurveNames}),
1537         implicitlyCA   NULL,
1538         curveName      PrintableString
1539     }
```

1540 Edwards EC private keys only support the use of the **curveName** selection to specify a curve name as
 1541 defined in [RFC7748] and the use of the **oId** selection to specify a curve through an ECDH algorithm as
 1542 defined in [RFC 8410]. Note that keys defined by RFC 7748 and RFC 8410 are incompatible.

1543 Note that when generating a Montgomery EC private key, the EC domain parameters are *not* specified in
 1544 the key's template. This is because Montgomery EC private keys are only generated as part of a
 1545 Montgomery EC key pair, and the EC domain parameters for the pair are specified in the template for the
 1546 Montgomery EC public key.

1547 The following is a sample template for creating a Montgomery EC private key object:

```
1548     CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
1549     CK_KEY_TYPE keyType = CKK_EC;
1550     CK_UTF8CHAR label[] = "A Montgomery EC private key object";
1551     CK_BYTE subject[] = {...};
1552     CK_BYTE id[] = {123};
1553     CK_BYTE ecParams[] = {...};
1554     CK_BYTE value[] = {...};
1555     CK_BBOOL true = CK_TRUE;
1556     CK_ATTRIBUTE template[] = {
1557         {CKA_CLASS, &class, sizeof(class)},
1558         {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
1559         {CKA_TOKEN, &true, sizeof(true)},
1560         {CKA_LABEL, label, sizeof(label)-1},
1561         {CKA_SUBJECT, subject, sizeof(subject)},
1562         {CKA_ID, id, sizeof(id)},
1563         {CKA_SENSITIVE, &true, sizeof(true)},
1564         {CKA_DERIVE, &true, sizeof(true)},
1565         {CKA_VALUE, value, sizeof(value)}
1566     };
```

1567 2.3.9 Elliptic curve key pair generation

1568 The EC (also related to ECDSA) key pair generation mechanism, denoted CKM_EC_KEY_PAIR_GEN, is
 1569 a key pair generation mechanism that uses the method defined by the ANSI X9.62 and X9.63 standards.

1570 The EC (also related to ECDSA) key pair generation mechanism, denoted
1571 CKM_EC_KEY_PAIR_GEN_W_EXTRA_BITS, is a key pair generation mechanism that uses the method
1572 defined by FIPS 186-4 Appendix B.4.1.
1573 These mechanisms do not have a parameter.
1574 These mechanisms generate EC public/private key pairs with particular EC domain parameters, as
1575 specified in the **CKA_EC_PARAMS** attribute of the template for the public key. Note that this version of
1576 Cryptoki does not include a mechanism for generating these EC domain parameters.
1577 These mechanism contribute the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_EC_POINT** attributes to the
1578 new public key and the **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_EC_PARAMS** and **CKA_VALUE**
1579 attributes to the new private key. Other attributes supported by the EC public and private key types
1580 (specifically, the flags indicating which functions the keys support) may also be specified in the templates
1581 for the keys, or else are assigned default initial values.
1582 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1583 specify the minimum and maximum supported number of bits in the field sizes, respectively. For
1584 example, if a Cryptoki library supports only ECDSA using a field of characteristic 2 which has between
1585 2^{200} and 2^{300} elements, then *ulMinKeySize* = 201 and *ulMaxKeySize* = 301 (when written in binary
1586 notation, the number 2^{200} consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number.
1587 Similarly, 2^{300} is a 301-bit number).

1588 2.3.10 Edwards Elliptic curve key pair generation

1589 The Edwards EC key pair generation mechanism, denoted **CKM_EC_EDWARDS_KEY_PAIR_GEN**, is a
1590 key pair generation mechanism for EC keys over curves represented in Edwards form.

1591 This mechanism does not have a parameter.

1592 The mechanism can only generate EC public/private key pairs over the curves edwards25519 and
1593 edwards448 as defined in RFC 8032 or the curves id-Ed25519 and id-Ed448 as defined in RFC 8410.
1594 These curves can only be specified in the **CKA_EC_PARAMS** attribute of the template for the public key
1595 using the **curveName** or the **old** methods. Attempts to generate keys over these curves using any other
1596 EC key pair generation mechanism will fail with **CKR_CURVE_NOT_SUPPORTED**.

1597 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_EC_POINT** attributes to the
1598 new public key and the **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_EC_PARAMS** and **CKA_VALUE**
1599 attributes to the new private key. Other attributes supported by the Edwards EC public and private key
1600 types (specifically, the flags indicating which functions the keys support) may also be specified in the
1601 templates for the keys, or else are assigned default initial values.

1602 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1603 specify the minimum and maximum supported number of bits in the field sizes, respectively. For this
1604 mechanism, the only allowed values are 255 and 448 as RFC 8032 only defines curves of these two
1605 sizes. A Cryptoki implementation may support one or both of these curves and should set the
1606 *ulMinKeySize* and *ulMaxKeySize* fields accordingly.

1607 2.3.11 Montgomery Elliptic curve key pair generation

1608 The Montgomery EC key pair generation mechanism, denoted
1609 **CKM_EC_MONTGOMERY_KEY_PAIR_GEN**, is a key pair generation mechanism for EC keys over
1610 curves represented in Montgomery form.

1611 This mechanism does not have a parameter.

1612 The mechanism can only generate Montgomery EC public/private key pairs over the curves curve25519
1613 and curve448 as defined in RFC 7748 or the curves id-X25519 and id-X448 as defined in RFC 8410.
1614 These curves can only be specified in the **CKA_EC_PARAMS** attribute of the template for the public key
1615 using the **curveName** or **old** methods. Attempts to generate keys over these curves using any other EC
1616 key pair generation mechanism will fail with **CKR_CURVE_NOT_SUPPORTED**.

1617 The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_EC_POINT** attributes to the
1618 new public key and the **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_EC_PARAMS** and **CKA_VALUE**

1619 attributes to the new private key. Other attributes supported by the EC public and private key types
1620 (specifically, the flags indicating which functions the keys support) may also be specified in the templates
1621 for the keys, or else are assigned default initial values.

1622 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1623 specify the minimum and maximum supported number of bits in the field sizes, respectively. For this
1624 mechanism, the only allowed values are 255 and 448 as RFC 7748 only defines curves of these two
1625 sizes. A Cryptoki implementation may support one or both of these curves and should set the
1626 *ulMinKeySize* and *ulMaxKeySize* fields accordingly.

1627 **2.3.12 ECDSA without hashing**

1628 Refer section 2.3.1 for signature encoding.

1629 The ECDSA without hashing mechanism, denoted **CKM_ECDSA**, is a mechanism for single-part
1630 signatures and verification for ECDSA. (This mechanism corresponds only to the part of ECDSA that
1631 processes the hash value, which should not be longer than 1024 bits; it does not compute the hash
1632 value.)

1633 This mechanism does not have a parameter.

1634 Constraints on key types and the length of data are summarized in the following table:

1635 *Table 40, ECDSA without hashing: Key and Data Length*

Function	Key type	Input length	Output length
C_Sign ¹	ECDSA private key	any ³	2nLen
C_Verify ¹	ECDSA public key	any ³ , ≤2nLen ²	N/A

1636 ¹ Single-part operations only.

1637 ² Data length, signature length.

1638 ³ Input the entire raw digest. Internally, this will be truncated to the appropriate number of bits.

1639 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1640 specify the minimum and maximum supported number of bits in the field sizes, respectively. For
1641 example, if a Cryptoki library supports only ECDSA using a field of characteristic 2 which has between
1642 2^{200} and 2^{300} elements (inclusive), then *ulMinKeySize* = 201 and *ulMaxKeySize* = 301 (when written in
1643 binary notation, the number 2^{200} consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number.
1644 Similarly, 2^{300} is a 301-bit number).

1645 **2.3.13 ECDSA with hashing**

1646 Refer to section 2.3.1 for signature encoding.

1647 The ECDSA with SHA-1, SHA-224, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512
1648 mechanism, denoted

1649 **CKM_ECDSA_[SHA1|SHA224|SHA384|SHA512|SHA3_224|SHA3_256|SHA3_384|SHA3_512]**

1650 respectively, is a mechanism for single- and multiple-part signatures and verification for ECDSA. This
1651 mechanism computes the entire ECDSA specification, including the hashing with SHA-1, SHA-224, SHA-
1652 384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512 respectively.

1653 This mechanism does not have a parameter.

1654 Constraints on key types and the length of data are summarized in the following table:

1655 *Table 41, ECDSA with hashing: Key and Data Length*

Function	Key type	Input length	Output length
C_Sign	ECDSA private key	any	2nLen
C_Verify	ECDSA public key	any, ≤2nLen ²	N/A

1656 ² Data length, signature length.

1657 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 1658 specify the minimum and maximum supported number of bits in the field sizes, respectively. For
 1659 example, if a Cryptoki library supports only ECDSA using a field of characteristic 2 which has between
 1660 2^{200} and 2^{300} elements, then *ulMinKeySize* = 201 and *ulMaxKeySize* = 301 (when written in binary
 1661 notation, the number 2^{200} consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number.
 1662 Similarly, 2^{300} is a 301-bit number).

1663 2.3.14 EdDSA

1664 The EdDSA mechanism, denoted **CKM_EDDSA**, is a mechanism for single-part and multipart signatures
 1665 and verification for EdDSA. This mechanism implements the five EdDSA signature schemes defined in
 1666 RFC 8032 and RFC 8410.

1667 For curves according to RFC 8032, this mechanism has an optional parameter, a **CK_EDDSA_PARAMS**
 1668 structure. The absence or presence of the parameter as well as its content is used to identify which
 1669 signature scheme is to be used. The following table enumerates the five signature schemes defined in
 1670 RFC 8032 and all supported permutations of the mechanism parameter and its content.

1671 *Table 42, Mapping to RFC 8032 Signature Schemes*

Signature Scheme	Mechanism Param	phFlag	Context Data
Ed25519	<i>Not Required</i>	N/A	N/A
Ed25519ctx	<i>Required</i>	False	Optional
Ed25519ph	<i>Required</i>	True	Optional
Ed448	<i>Required</i>	False	Optional
Ed448ph	<i>Required</i>	True	Optional

1672 For curves according to RFC 8410, the mechanism is implicitly given by the curve, which is EdDSA in
 1673 pure mode.

1674 Constraints on key types and the length of data are summarized in the following table:

1675 *Table 43, EdDSA: Key and Data Length*

Function	Key type	Input length	Output length
C_Sign	<i>CKK_EC_EDWARDS private key</i>	any	$2bLen$
C_Verify	<i>CKK_EC_EDWARDS public key</i>	any, $\leq 2bLen^2$	N/A

1676 ² Data length, signature length.

1677 Note that for EdDSA in pure mode, Ed25519 and Ed448 the data must be processed twice. Therefore, a
 1678 token might need to cache all the data, especially when used with C_SignUpdate/C_VerifyUpdate. If
 1679 tokens are unable to do so they can return CKR_TOKEN_RESOURCE_EXCEEDED.

1680 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
 1681 specify the minimum and maximum supported number of bits in the field sizes, respectively. For this
 1682 mechanism, the only allowed values are 255 and 448 as RFC 8032 and RFC 8410 only define curves of
 1683 these two sizes. A Cryptoki implementation may support one or both of these curves and should set the
 1684 *ulMinKeySize* and *ulMaxKeySize* fields accordingly.

1685 2.3.15 XEdDSA

1686 The XEdDSA mechanism, denoted **CKM_XEDDSA**, is a mechanism for single-part signatures and
 1687 verification for XEdDSA. This mechanism implements the XEdDSA signature scheme defined in
 1688 **[XEDDSA]**. CKM_XEDDSA operates on CKK_EC_MONTGOMERY type EC keys, which allows these

1689 keys to be used both for signing/verification and for Diffie-Hellman style key-exchanges. This double use
1690 is necessary for the Extended Triple Diffie-Hellman where the long-term identity key is used to sign short-
1691 term keys and also contributes to the DH key-exchange.

1692 This mechanism has a parameter, a **CK_XEDDSA_PARAMS** structure.

1693 *Table 44, XEdDSA: Key and Data Length*

Function	Key type	Input length	Output length
C_Sign ¹	CKK_EC_MONTGOMERY private	any ³	2b
C_Verify ¹	CKK_EC_MONTGOMERY public	any ³ , ≤2b ²	N/A

1694 ² Data length, signature length.

1695 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure
1696 specify the minimum and maximum supported number of bits in the field sizes, respectively. For this
1697 mechanism, the only allowed values are 255 and 448 as [XEDDSA] only defines curves of these two
1698 sizes. A Cryptoki implementation may support one or both of these curves and should set the
1699 *ulMinKeySize* and *ulMaxKeySize* fields accordingly.

1700 2.3.16 EC mechanism parameters

1701 ♦ **CK_EDDSA_PARAMS, CK_EDDSA_PARAMS_PTR**

1702 **CK_EDDSA_PARAMS** is a structure that provides the parameters for the **CKM_EDDSA** signature
1703 mechanism. The structure is defined as follows:

```
1704     typedef struct CK_EDDSA_PARAMS {  
1705         CK_BBOOL      phFlag;  
1706         CK ULONG      ulContextDataLen;  
1707         CK_BYTE_PTR   pContextData;  
1708     } CK_EDDSA_PARAMS;
```

1709

1710 The fields of the structure have the following meanings:

1711 *phFlag* a Boolean value which indicates if Prehashed variant of EdDSA should
1712 be used

1713 *ulContextDataLen* the length in bytes of the context data where 0 <= *ulContextDataLen* <=
1714 255.

1715 *pContextData* context data shared between the signer and verifier

1716 **CK_EDDSA_PARAMS_PTR** is a pointer to a **CK_EDDSA_PARAMS**.

1717

1718 ♦ **CK_XEDDSA_PARAMS, CK_XEDDSA_PARAMS_PTR**

1719 **CK_XEDDSA_PARAMS** is a structure that provides the parameters for the **CKM_XEDDSA** signature
1720 mechanism. The structure is defined as follows:

```

1721     typedef struct CK_XEDDSA_PARAMS {
1722         CK_XEDDSA_HASH_TYPE hash;
1723     } CK_XEDDSA_PARAMS;
1724
1725     The fields of the structure have the following meanings:
1726             hash      a Hash mechanism to be used by the mechanism.
1727
1728     CK_XEDDSA_PARAMS_PTR is a pointer to a CK_XEDDSA_PARAMS.
1729
1730     ♦ CK_XEDDSA_HASH_TYPE, CK_XEDDSA_HASH_TYPE_PTR
1731
1732     CK_XEDDSA_HASH_TYPE is used to indicate the hash function used in XEDDSA. It is defined as
1733     follows:
1734         typedef CK ULONG CK_XEDDSA_HASH_TYPE;
1735
1736     The following table lists the defined functions.
1737
1738     Table 45, EC: Key Derivation Functions

```

Source Identifier
CKM_BLAKE2B_256
CKM_BLAKE2B_512
CKM_SHA3_256
CKM_SHA3_512
CKM_SHA256
CKM_SHA512

```

1739
1740     CK_XEDDSA_HASH_TYPE_PTR is a pointer to a CK_XEDDSA_HASH_TYPE.
1741
1742     ♦ CK_EC_KDF_TYPE, CK_EC_KDF_TYPE_PTR
1743
1744     CK_EC_KDF_TYPE is used to indicate the Key Derivation Function (KDF) applied to derive keying data
1745     from a shared secret. The key derivation function will be used by the EC key agreement schemes. It is
1746     defined as follows:
1747         typedef CK ULONG CK_EC_KDF_TYPE;
1748
1749     The following table lists the defined functions.
1750
1751     Table 46, EC: Key Derivation Functions

```

Source Identifier
CKD_NULL
CKD_SHA1_KDF
CKD_SHA224_KDF
CKD_SHA256_KDF
CKD_SHA384_KDF
CKD_SHA512_KDF
CKD_SHA3_224_KDF

CKD_SHA3_256_KDF
CKD_SHA3_384_KDF
CKD_SHA3_512_KDF
CKD_SHA1_KDF_SP800
CKD_SHA224_KDF_SP800
CKD_SHA256_KDF_SP800
CKD_SHA384_KDF_SP800
CKD_SHA512_KDF_SP800
CKD_SHA3_224_KDF_SP800
CKD_SHA3_256_KDF_SP800
CKD_SHA3_384_KDF_SP800
CKD_SHA3_512_KDF_SP800
CKD_BLAKE2B_160_KDF
CKD_BLAKE2B_256_KDF
CKD_BLAKE2B_384_KDF
CKD_BLAKE2B_512_KDF

- 1747 The key derivation function **CKD_NULL** produces a raw shared secret value without applying any key
 1748 derivation function.
- 1749 The key derivation functions
 1750 **CKD_[SHA1|SHA224|SHA384|SHA512|SHA3_224|SHA3_256|SHA3_384|SHA3_512]_KDF**, which are
 1751 based on SHA-1, SHA-224, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512
 1752 respectively, derive keying data from the shared secret value as defined in [ANSI X9.63].
- 1753 The key derivation functions
 1754 **CKD_[SHA1|SHA224|SHA384|SHA512|SHA3_224|SHA3_256|SHA3_384|SHA3_512]_KDF_SP800**,
 1755 which are based on SHA-1, SHA-224, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512
 1756 respectively, derive keying data from the shared secret value as defined in [FIPS SP800-56A] section
 1757 5.8.1.1.
- 1758 The key derivation functions **CKD_BLAKE2B_[160|256|384|512]_KDF**, which are based on the Blake2b
 1759 family of hashes, derive keying data from the shared secret value as defined in [FIPS SP800-56A] section
 1760 5.8.1.1. **CK_EC_KDF_TYPE_PTR** is a pointer to a **CK_EC_KDF_TYPE**.
- 1761
- 1762 ◆ **CK_ECDH1_DERIVE_PARAMS**, **CK_ECDH1_DERIVE_PARAMS_PTR**
- 1763 **CK_ECDH1_DERIVE_PARAMS** is a structure that provides the parameters for the
 1764 **CKM_ECDH1_DERIVE** and **CKM_ECDH1_COFACTOR_DERIVE** key derivation mechanisms, where
 1765 each party contributes one key pair. The structure is defined as follows:
- ```
1766 typedef struct CK_ECDH1_DERIVE_PARAMS {
1767 CK_EC_KDF_TYPE kdf;
1768 CK ULONG ulSharedDataLen;
1769 CK BYTE PTR pSharedData;
1770 CK ULONG ulPublicDataLen;
1771 CK BYTE PTR pPublicData;
1772 } CK_ECDH1_DERIVE_PARAMS;
```
- 1773
- 1774 The fields of the structure have the following meanings:
- 1775                    *kdf*      *key derivation function used on the shared secret value*

|      |                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1776 | <i>ulSharedDataLen</i>                                                                                                                                        | <i>the length in bytes of the shared info</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 1777 | <i>pSharedData</i>                                                                                                                                            | <i>some data shared between the two parties</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1778 | <i>ulPublicDataLen</i>                                                                                                                                        | <i>the length in bytes of the other party's EC public key</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 1779 | <i>pPublicData</i> <sup>1</sup>                                                                                                                               | <i>pointer to other party's EC public key value. A token MUST be able to accept this value encoded as a raw octet string (as per section A.5.2 of [ANSI X9.62]). A token MAY, in addition, support accepting this value as a DER-encoded ECPoint (as per section E.6 of [ANSI X9.62]) i.e. the same as a CKA_EC_POINT encoding. The calling application is responsible for converting the offered public key to the compressed or uncompressed forms of these encodings if the token does not support the offered form.</i> |
| 1780 |                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1781 |                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1782 |                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1783 |                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1784 |                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1785 |                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1786 |                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1787 | With the key derivation function <b>CKD_NULL</b> , <i>pSharedData</i> must be NULL and <i>ulSharedDataLen</i> must be zero. With the key derivation functions |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1788 |                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1789 | <b>CKD_[SHA1 SHA224 SHA384 SHA512]_KDF</b> ,                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1790 | <b>CKD_[SHA1 SHA224 SHA384 SHA512]_KDF_SP800</b> , an                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1791 | optional <i>pSharedData</i> may be supplied, which consists of some data shared by the two parties intending                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1792 | to share the shared secret. Otherwise, <i>pSharedData</i> must be NULL and <i>ulSharedDataLen</i> must be zero.                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1793 | <b>CK_ECDH1_DERIVE_PARAMS_PTR</b> is a pointer to a <b>CK_ECDH1_DERIVE_PARAMS</b> .                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1794 | ◆ <b>CK_ECDH2_DERIVE_PARAMS</b> , <b>CK_ECDH2_DERIVE_PARAMS_PTR</b>                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1795 | <b>CK_ECDH2_DERIVE_PARAMS</b> is a structure that provides the parameters to the                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1796 | <b>CKM_ECMQV_DERIVE</b> key derivation mechanism, where each party contributes two key pairs. The                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1797 | structure is defined as follows:                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1798 | typedef struct CK_ECDH2_DERIVE_PARAMS {                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1799 | CK_EC_KDF_TYPE kdf;                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1800 | CK ULONG ulSharedDataLen;                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1801 | CK_BYTE_PTR pSharedData;                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1802 | CK ULONG ulPublicDataLen;                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1803 | CK_BYTE_PTR pPublicData;                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1804 | CK ULONG ulPrivateDataLen;                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1805 | CK_OBJECT_HANDLE hPrivateKey;                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1806 | CK ULONG ulPublicDataLen2;                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1807 | CK_BYTE_PTR pPublicData2;                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1808 | } CK_ECDH2_DERIVE_PARAMS;                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1809 |                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1810 | The fields of the structure have the following meanings:                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1811 | <i>kdf</i>                                                                                                                                                    | <i>key derivation function used on the shared secret value</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 1812 | <i>ulSharedDataLen</i>                                                                                                                                        | <i>the length in bytes of the shared info</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 1813 | <i>pSharedData</i>                                                                                                                                            | <i>some data shared between the two parties</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 1814 | <i>ulPublicDataLen</i>                                                                                                                                        | <i>the length in bytes of the other party's first EC public key</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

1. The encoding in V2.20 was not specified and resulted in different implementations choosing different encodings. Applications relying only on a V2.20 encoding (e.g. the DER variant) other than the one specified now (raw) may not work with all V2.30 compliant tokens.

1815                    *pPublicData*      pointer to other party's first EC public key value. Encoding rules are  
 1816                    as per *pPublicData* of CK\_ECDH1\_DERIVE\_PARAMS  
 1817                    *ulPrivateDataLen*     the length in bytes of the second EC private key  
 1818                    *hPrivateKeyData*     key handle for second EC private key  
 1819                    *ulPublicDataLen2*    the length in bytes of the other party's second EC public key  
 1820                    *pPublicData2*      pointer to other party's second EC public key value. Encoding rules  
 1821                    are as per *pPublicData* of CK\_ECDH1\_DERIVE\_PARAMS

1822 With the key derivation function **CKD\_NULL**, *pSharedData* must be NULL and *ulSharedDataLen* must be zero. With the key derivation function **CKD\_SHA1\_KDF**, an optional *pSharedData* may be supplied, which consists of some data shared by the two parties intending to share the shared secret. Otherwise, *pSharedData* must be NULL and *ulSharedDataLen* must be zero.  
 1823

1824       **CK\_ECDH2\_DERIVE\_PARAMS\_PTR** is a pointer to a **CK\_ECDH2\_DERIVE\_PARAMS**.  
 1825

1827

1828       ◆ **CK\_ECMQV\_DERIVE\_PARAMS, CK\_ECMQV\_DERIVE\_PARAMS\_PTR**

1829       **CK\_ECMQV\_DERIVE\_PARAMS** is a structure that provides the parameters to the  
 1830       **CKM\_ECMQV\_DERIVE** key derivation mechanism, where each party contributes two key pairs. The  
 1831       structure is defined as follows:

```

 1832 typedef struct CK_ECMQV_DERIVE_PARAMS {
 1833 CK_EC_KDF_TYPE kdf;
 1834 CK ULONG ulSharedDataLen;
 1835 CK_BYTE_PTR pSharedData;
 1836 CK ULONG ulPublicDataLen;
 1837 CK_BYTE_PTR pPublicData;
 1838 CK ULONG ulPrivateDataLen;
 1839 CK_OBJECT_HANDLE hPrivateKeyData;
 1840 CK ULONG ulPublicDataLen2;
 1841 CK_BYTE_PTR pPublicData2;
 1842 CK_OBJECT_HANDLE publicKey;
 1843 } CK_ECMQV_DERIVE_PARAMS;
 1844

```

1845       The fields of the structure have the following meanings:

|                              |                                                                                                                                           |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| 1846 <i>kdf</i>              | <i>key derivation function used on the shared secret value</i>                                                                            |
| 1847 <i>ulSharedDataLen</i>  | <i>the length in bytes of the shared info</i>                                                                                             |
| 1848 <i>pSharedData</i>      | <i>some data shared between the two parties</i>                                                                                           |
| 1849 <i>ulPublicDataLen</i>  | <i>the length in bytes of the other party's first EC public key</i>                                                                       |
| 1850 <i>pPublicData</i>      | <i>pointer to other party's first EC public key value. Encoding rules are</i><br>1851 <i>as per pPublicData of CK_ECDH1_DERIVE_PARAMS</i> |
| 1852 <i>ulPrivateDataLen</i> | <i>the length in bytes of the second EC private key</i>                                                                                   |
| 1853 <i>hPrivateKeyData</i>  | <i>key handle for second EC private key value</i>                                                                                         |

|      |                                                                                                                                                               |                                                                                                                             |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| 1854 | <i>ulPublicDataLen2</i>                                                                                                                                       | <i>the length in bytes of the other party's second EC public key</i>                                                        |
| 1855 | <i>pPublicData2</i>                                                                                                                                           | <i>pointer to other party's second EC public key value. Encoding rules are as per pPublicData of CK_ECDH1_DERIVE_PARAMS</i> |
| 1856 |                                                                                                                                                               |                                                                                                                             |
| 1857 | <i>publicKey</i>                                                                                                                                              | <i>Handle to the first party's ephemeral public key</i>                                                                     |
| 1858 | With the key derivation function <b>CKD_NULL</b> , <i>pSharedData</i> must be NULL and <i>ulSharedDataLen</i> must be zero. With the key derivation functions |                                                                                                                             |
| 1859 | <b>CKD_[SHA1 SHA224 SHA384 SHA512 SHA3_224 SHA3_256 SHA3_384 SHA3_512]_KDF</b> ,                                                                              |                                                                                                                             |
| 1860 | <b>CKD_[SHA1 SHA224 SHA384 SHA512 SHA3_224 SHA3_256 SHA3_384 SHA3_512]_KDF_SP800</b> , an                                                                     |                                                                                                                             |
| 1861 | optional <i>pSharedData</i> may be supplied, which consists of some data shared by the two parties intending                                                  |                                                                                                                             |
| 1862 | to share the shared secret. Otherwise, <i>pSharedData</i> must be NULL and <i>ulSharedDataLen</i> must be zero.                                               |                                                                                                                             |
| 1863 |                                                                                                                                                               |                                                                                                                             |
| 1864 | <b>CK_ECMQV_DERIVE_PARAMS_PTR</b> is a pointer to a <b>CK_ECMQV_DERIVE_PARAMS</b> .                                                                           |                                                                                                                             |
| 1865 | <h3>2.3.17 Elliptic curve Diffie-Hellman key derivation</h3>                                                                                                  |                                                                                                                             |
| 1866 | The elliptic curve Diffie-Hellman (ECDH) key derivation mechanism, denoted <b>CKM_ECDH1_DERIVE</b> , is a                                                     |                                                                                                                             |
| 1867 | mechanism for key derivation based on the Diffie-Hellman version of the elliptic curve key agreement                                                          |                                                                                                                             |
| 1868 | scheme, as defined in ANSI X9.63, where each party contributes one key pair all using the same EC                                                             |                                                                                                                             |
| 1869 | domain parameters.                                                                                                                                            |                                                                                                                             |
| 1870 | It has a parameter, a <b>CK_ECDH1_DERIVE_PARAMS</b> structure.                                                                                                |                                                                                                                             |
| 1871 | This mechanism derives a secret value, and truncates the result according to the <b>CKA_KEY_TYPE</b>                                                          |                                                                                                                             |
| 1872 | attribute of the template and, if it has one and the key type supports it, the <b>CKA_VALUE_LEN</b> attribute of                                              |                                                                                                                             |
| 1873 | the template. (The truncation removes bytes from the leading end of the secret value.) The mechanism                                                          |                                                                                                                             |
| 1874 | contributes the result as the <b>CKA_VALUE</b> attribute of the new key; other attributes required by the key                                                 |                                                                                                                             |
| 1875 | type must be specified in the template.                                                                                                                       |                                                                                                                             |
| 1876 | This mechanism has the following rules about key sensitivity and extractability:                                                                              |                                                                                                                             |
| 1877 | • The <b>CKA_SENSITIVE</b> and <b>CKA_EXTRACTABLE</b> attributes in the template for the new key can both                                                     |                                                                                                                             |
| 1878 | be specified to be either CK_TRUE or CK_FALSE. If omitted, these attributes each take on some                                                                 |                                                                                                                             |
| 1879 | default value.                                                                                                                                                |                                                                                                                             |
| 1880 | • If the base key has its <b>CKA_ALWAYS_SENSITIVE</b> attribute set to CK_FALSE, then the derived key                                                         |                                                                                                                             |
| 1881 | will as well. If the base key has its <b>CKA_ALWAYS_SENSITIVE</b> attribute set to CK_TRUE, then the                                                          |                                                                                                                             |
| 1882 | derived key has its <b>CKA_ALWAYS_SENSITIVE</b> attribute set to the same value as its                                                                        |                                                                                                                             |
| 1883 | <b>CKA_SENSITIVE</b> attribute.                                                                                                                               |                                                                                                                             |
| 1884 | • Similarly, if the base key has its <b>CKA_NEVER_EXTRACTABLE</b> attribute set to CK_FALSE, then the                                                         |                                                                                                                             |
| 1885 | derived key will, too. If the base key has its <b>CKA_NEVER_EXTRACTABLE</b> attribute set to                                                                  |                                                                                                                             |
| 1886 | CK_TRUE, then the derived key has its <b>CKA_NEVER_EXTRACTABLE</b> attribute set to the <i>opposite</i>                                                       |                                                                                                                             |
| 1887 | value from its <b>CKA_EXTRACTABLE</b> attribute.                                                                                                              |                                                                                                                             |
| 1888 | For this mechanism, the <i>ulMinKeySize</i> and <i>ulMaxKeySize</i> fields of the <b>CK_MECHANISM_INFO</b> structure                                          |                                                                                                                             |
| 1889 | specify the minimum and maximum supported number of bits in the field sizes, respectively. For                                                                |                                                                                                                             |
| 1890 | example, if a Cryptoki library supports only EC using a field of characteristic 2 which has between $2^{200}$                                                 |                                                                                                                             |
| 1891 | and $2^{300}$ elements, then <i>ulMinKeySize</i> = 201 and <i>ulMaxKeySize</i> = 301 (when written in binary notation,                                        |                                                                                                                             |
| 1892 | the number $2^{200}$ consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number. Similarly, $2^{300}$                                       |                                                                                                                             |
| 1893 | is a 301-bit number).                                                                                                                                         |                                                                                                                             |
| 1894 | Constraints on key types are summarized in the following table:                                                                                               |                                                                                                                             |

1895 Table 47: ECDH: Allowed Key Types

| Function | Key type                    |
|----------|-----------------------------|
| C_Derive | CKK_EC or CKK_EC_MONTGOMERY |

### 1896 2.3.18 Elliptic curve Diffie-Hellman with cofactor key derivation

1897 The elliptic curve Diffie-Hellman (ECDH) with cofactor key derivation mechanism, denoted  
1898 **CKM\_ECDH1\_COFACTOR\_DERIVE**, is a mechanism for key derivation based on the cofactor Diffie-  
1899 Hellman version of the elliptic curve key agreement scheme, as defined in ANSI X9.63, where each party  
1900 contributes one key pair all using the same EC domain parameters. Cofactor multiplication is  
1901 computationally efficient and helps to prevent security problems like small group attacks.

1902 It has a parameter, a **CK\_ECDH1\_DERIVE\_PARAMS** structure.

1903 This mechanism derives a secret value, and truncates the result according to the **CKA\_KEY\_TYPE**  
1904 attribute of the template and, if it has one and the key type supports it, the **CKA\_VALUE\_LEN** attribute of  
1905 the template. (The truncation removes bytes from the leading end of the secret value.) The mechanism  
1906 contributes the result as the **CKA\_VALUE** attribute of the new key; other attributes required by the key  
1907 type must be specified in the template.

1908 This mechanism has the following rules about key sensitivity and extractability:

- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
1910 be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
1911 default value.
- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
1913 will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the  
1914 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
1915 **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
1917 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
1918 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
1919 value from its **CKA\_EXTRACTABLE** attribute.

1920 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
1921 specify the minimum and maximum supported number of bits in the field sizes, respectively. For  
1922 example, if a Cryptoki library supports only EC using a field of characteristic 2 which has between  $2^{200}$   
1923 and  $2^{300}$  elements, then *ulMinKeySize* = 201 and *ulMaxKeySize* = 301 (when written in binary notation,  
1924 the number  $2^{200}$  consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number. Similarly,  $2^{300}$   
1925 is a 301-bit number).

1926 Constraints on key types are summarized in the following table:

1927 Table 48: ECDH with cofactor: Allowed Key Types

| Function | Key type |
|----------|----------|
| C_Derive | CKK_EC   |

### 1928 2.3.19 Elliptic curve Menezes-Qu-Vanstone key derivation

1929 The elliptic curve Menezes-Qu-Vanstone (ECMQV) key derivation mechanism, denoted  
1930 **CKM\_ECMQV\_DERIVE**, is a mechanism for key derivation based the MQV version of the elliptic curve  
1931 key agreement scheme, as defined in ANSI X9.63, where each party contributes two key pairs all using  
1932 the same EC domain parameters.

1933 It has a parameter, a **CK\_ECMQV\_DERIVE\_PARAMS** structure.

1934 This mechanism derives a secret value, and truncates the result according to the **CKA\_KEY\_TYPE**  
1935 attribute of the template and, if it has one and the key type supports it, the **CKA\_VALUE\_LEN** attribute of  
1936 the template. (The truncation removes bytes from the leading end of the secret value.) The mechanism

1937 contributes the result as the **CKA\_VALUE** attribute of the new key; other attributes required by the key  
 1938 type must be specified in the template.  
 1939 This mechanism has the following rules about key sensitivity and extractability:  
 1940 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
 1941 be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
 1942 default value.  
 1943 • If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
 1944 will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the  
 1945 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
 1946 **CKA\_SENSITIVE** attribute.  
 1947 • Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
 1948 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
 1949 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
 1950 value from its **CKA\_EXTRACTABLE** attribute.  
 1951 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 1952 specify the minimum and maximum supported number of bits in the field sizes, respectively. For  
 1953 example, if a Cryptoki library supports only EC using a field of characteristic 2 which has between  $2^{200}$   
 1954 and  $2^{300}$  elements, then *ulMinKeySize* = 201 and *ulMaxKeySize* = 301 (when written in binary notation,  
 1955 the number  $2^{200}$  consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number. Similarly,  $2^{300}$   
 1956 is a 301-bit number).  
 1957 Constraints on key types are summarized in the following table:  
 1958 *Table 49: ECDH MQV: Allowed Key Types*

| Function | Key type |
|----------|----------|
| C_Derive | CKK_EC   |

1959 **2.3.20 ECDH AES KEY WRAP**  
 1960 The ECDH AES KEY WRAP mechanism, denoted **CKM\_ECDH\_AES\_KEY\_WRAP**, is a mechanism  
 1961 based on elliptic curve public-key crypto-system and the AES key wrap mechanism. It supports single-  
 1962 part key wrapping; and key unwrapping.  
 1963 It has a parameter, a **CK\_ECDH\_AES\_KEY\_WRAP\_PARAMS** structure.  
 1964  
 1965 The mechanism can wrap and unwrap an asymmetric target key of any length and type using an EC  
 1966 key.  
 1967     - A temporary AES key is derived from a temporary EC key and the wrapping EC key  
 1968         using the **CKM\_ECDH1\_DERIVE** mechanism.  
 1969     - The derived AES key is used for wrapping the target key using the  
 1970         **CKM\_AES\_KEY\_WRAP\_KWP** mechanism.  
 1971  
 1972 For wrapping, the mechanism -  
 1973     • Generates a temporary random EC key (transport key) having the same parameters as the  
 1974         wrapping EC key (and domain parameters). Saves the transport key public key material.  
 1975     • Performs ECDH operation using **CKM\_ECDH1\_DERIVE** with parameters of kdf,  
 1976         
 1977         key of wrapping EC key and gets the first ulAESKeyBits bits of the derived key to be the  
 1978         temporary AES key.  
 1979     • Wraps the target key with the temporary AES key using **CKM\_AES\_KEY\_WRAP\_KWP** ([AES  
 1980         KEYWRAP] section 6.3).  
 1981     • Zeroizes the temporary AES key and EC transport private key.

- 1982     • Concatenates public key material of the transport key and output the concatenated blob. The first  
 1983       part is the public key material of the transport key and the second part is the wrapped target key.

1984  
 1985     The recommended format for an asymmetric target key being wrapped is as a PKCS8  
 1986       PrivateKeyInfo

1987  
 1988     The use of Attributes in the PrivateKeyInfo structure is OPTIONAL. In case of conflicts between the  
 1989       object attribute template, and Attributes in the PrivateKeyInfo structure, an error should be thrown.

1990  
 1991     For unwrapping, the mechanism -

- 1992     • Splits the input into two parts. The first part is the public key material of the transport key and the  
 1993       second part is the wrapped target key. The length of the first part is equal to the length of the  
 1994       public key material of the unwrapping EC key.

1995       *Note: since the transport key and the wrapping EC key share the same domain, the length of the  
 1996       public key material of the transport key is the same length of the public key material of the  
 1997       unwrapping EC key.*

- 1998     • Performs ECDH operation using **CKM\_ECDH1\_DERIVE** with parameters of kdf,  
 1999       ulSharedDataLen and pSharedData using the private part of unwrapping EC key and the public  
 2000       part of the transport EC key and gets first ulAESKeyBits bits of the derived key to be the  
 2001       temporary AES key.
- 2002     • Un-wraps the target key from the second part with the temporary AES key using  
 2003       **CKM\_AES\_KEY\_WRAP\_KWP** ([AES KEYWRAP] section 6.3).
- 2004     • Zeroizes the temporary AES key.

2005  
 2006     Table 50, CKM\_ECDH\_AES\_KEY\_WRAP Mechanisms vs. Functions

| Mechanism             | Functions         |               |                      |        |                    |               |        |
|-----------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                       | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_ECDH_AES_KEY_WRAP |                   |               |                      |        |                    | ✓             |        |

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

2007  
 2008     Constraints on key types are summarized in the following table:

2009     Table 51: ECDH AES Key Wrap: Allowed Key Types

| Function | Key type                    |
|----------|-----------------------------|
| C_Derive | CKK_EC or CKK_EC_MONTGOMERY |

## 2010     2.3.21 ECDH AES KEY WRAP mechanism parameters

- 2011     ♦ **CK\_ECDH\_AES\_KEY\_WRAP\_PARAMS; CK\_ECDH\_AES\_KEY\_WRAP\_PARAMS\_PTR**

2012     **CK\_ECDH\_AES\_KEY\_WRAP\_PARAMS** is a structure that provides the parameters to the  
 2013       **CKM\_ECDH\_AES\_KEY\_WRAP** mechanism. It is defined as follows:

2014  
 2015       typedef struct CK\_ECDH\_AES\_KEY\_WRAP\_PARAMS {  
 2016           CK ULONG                   ulAESKeyBits;

```

2017 CK_EC_KDF_TYPE kdf;
2018 CK ULONG ulSharedDataLen;
2019 CK_BYTE_PTR pSharedData;
2020 } CK_ECDH_AES_KEY_WRAP_PARAMS;
2021
2022 The fields of the structure have the following meanings:
2023
2024 ulAESKeyBits length of the temporary AES key in bits. Can be only 128, 192 or
2025 256.
2026 kdf key derivation function used on the shared secret value to generate
2027 AES key.
2028 ulSharedDataLen the length in bytes of the shared info
2029 pSharedData Some data shared between the two parties
2030
2031 CK_ECDH_AES_KEY_WRAP_PARAMS_PTR is a pointer to a
2032 CK_ECDH_AES_KEY_WRAP_PARAMS.
2033

```

### 2034 2.3.22 FIPS 186-4

2035 When CKM\_ECDSA is operated in FIPS mode, the curves SHALL either be NIST recommended curves  
 2036 (with a fixed set of domain parameters) or curves with domain parameters generated as specified by  
 2037 ANSI X9.64. The NIST recommended curves are:

```

2038
2039 P-192, P-224, P-256, P-384, P-521
2040 K-163, B-163, K-233, B-233
2041 K-283, B-283, K-409, B-409
2042 K-571, B-571

```

## 2043 2.4 Diffie-Hellman

2044 *Table 52, Diffie-Hellman Mechanisms vs. Functions*

| Mechanism                  | Functions         |               |                      |        |                    |               |        |
|----------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                            | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_DH_PKCS_KEY_PAIR_GEN   |                   |               |                      |        | ✓                  |               |        |
| CKM_DH_PKCS_PARAMETER_GEN  |                   |               |                      |        | ✓                  |               |        |
| CKM_DH_PKCS_DERIVE         |                   |               |                      |        |                    |               | ✓      |
| CKM_X9_42_DH_KEY_PAIR_GEN  |                   |               |                      |        | ✓                  |               |        |
| CKM_X9_42_DH_PARAMETER_GEN |                   |               |                      |        | ✓                  |               |        |
| CKM_X9_42_DH_DERIVE        |                   |               |                      |        |                    |               | ✓      |
| CKM_X9_42_DH_HYBRID_DERIVE |                   |               |                      |        |                    |               | ✓      |

| Mechanism            | Functions         |               |                      |        |                    |               |        |
|----------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                      | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_X9_42_MQV_DERIVE |                   |               |                      |        |                    |               | ✓      |

## 2.4.1 Definitions

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

Mechanisms:

- CKM\_DH\_PKCS\_KEY\_PAIR\_GEN
- CKM\_DH\_PKCS\_PARAMETER\_GEN
- CKM\_DH\_PKCS\_DERIVE
- CKM\_X9\_42\_DH\_KEY\_PAIR\_GEN
- CKM\_X9\_42\_DH\_PARAMETER\_GEN
- CKM\_X9\_42\_DH\_DERIVE
- CKM\_X9\_42\_DH\_HYBRID\_DERIVE
- CKM\_X9\_42\_MQV\_DERIVE

## 2.4.2 Diffie-Hellman public key objects

Diffie-Hellman public key objects (object class **CKO\_PUBLIC\_KEY**, key type **CKK\_DH**) hold Diffie-Hellman public keys. The following table defines the Diffie-Hellman public key object attributes, in addition to the common attributes defined for this object class:

Table 53, *Diffie-Hellman Public Key Object Attributes*

| Attribute                | Data type   | Meaning          |
|--------------------------|-------------|------------------|
| CKA_PRIME <sup>1,3</sup> | Big integer | Prime $p$        |
| CKA_BASE <sup>1,3</sup>  | Big integer | Base $g$         |
| CKA_VALUE <sup>1,4</sup> | Big integer | Public value $y$ |

- Refer to [PKCS11-Base] table 11 for footnotes

The **CKA\_PRIME** and **CKA\_BASE** attribute values are collectively the "Diffie-Hellman domain parameters". Depending on the token, there may be limits on the length of the key components. See PKCS #3 for more information on Diffie-Hellman keys.

The following is a sample template for creating a Diffie-Hellman public key object:

```

2068 CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
2069 CK_KEY_TYPE keyType = CKK_DH;
2070 CK_UTF8CHAR label[] = "A Diffie-Hellman public key object";
2071 CK_BYTE prime[] = {...};
2072 CK_BYTE base[] = {...};
2073 CK_BYTE value[] = {...};
2074 CK_BBOOL true = CK_TRUE;
2075 CK_ATTRIBUTE template[] = {
2076 {CKA_CLASS, &class, sizeof(class)},

```

```

2077 { CKA_KEY_TYPE, &keyType, sizeof(keyType) },
2078 { CKA_TOKEN, &true, sizeof(true) },
2079 { CKA_LABEL, label, sizeof(label)-1 },
2080 { CKA_PRIME, prime, sizeof(prime) },
2081 { CKA_BASE, base, sizeof(base) },
2082 { CKA_VALUE, value, sizeof(value) }
2083 } ;

```

## 2.4.3 X9.42 Diffie-Hellman public key objects

X9.42 Diffie-Hellman public key objects (object class **CKO\_PUBLIC\_KEY**, key type **CKK\_X9\_42\_DH**) hold X9.42 Diffie-Hellman public keys. The following table defines the X9.42 Diffie-Hellman public key object attributes, in addition to the common attributes defined for this object class:

2088 *Table 54, X9.42 Diffie-Hellman Public Key Object Attributes*

| Attribute                   | Data type   | Meaning                                             |
|-----------------------------|-------------|-----------------------------------------------------|
| CKA_PRIME <sup>1,3</sup>    | Big integer | Prime $p$ ( $\geq 1024$ bits, in steps of 256 bits) |
| CKA_BASE <sup>1,3</sup>     | Big integer | Base $g$                                            |
| CKA_SUBPRIME <sup>1,3</sup> | Big integer | Subprime $q$ ( $\geq 160$ bits)                     |
| CKA_VALUE <sup>1,4</sup>    | Big integer | Public value $y$                                    |

2089 - Refer to [PKCS11-Base] table 11 for footnotes

2090 The **CKA\_PRIME**, **CKA\_BASE** and **CKA\_SUBPRIME** attribute values are collectively the “X9.42 Diffie-  
2091 Hellman domain parameters”. See the ANSI X9.42 standard for more information on X9.42 Diffie-  
2092 Hellman keys.

2093 The following is a sample template for creating a X9.42 Diffie-Hellman public key object:

```

2094 CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
2095 CK_KEY_TYPE keyType = CKK_X9_42_DH;
2096 CK_UTF8CHAR label[] = "A X9.42 Diffie-Hellman public key
2097 object";
2098 CK_BYTE prime[] = {...};
2099 CK_BYTE base[] = {...};
2100 CK_BYTE subprime[] = {...};
2101 CK_BYTE value[] = {...};
2102 CK_BBOOL true = CK_TRUE;
2103 CK_ATTRIBUTE template[] = {
2104 { CKA_CLASS, &class, sizeof(class) },
2105 { CKA_KEY_TYPE, &keyType, sizeof(keyType) },
2106 { CKA_TOKEN, &true, sizeof(true) },
2107 { CKA_LABEL, label, sizeof(label)-1 },
2108 { CKA_PRIME, prime, sizeof(prime) },
2109 { CKA_BASE, base, sizeof(base) },
2110 { CKA_SUBPRIME, subprime, sizeof(subprime) },
2111 { CKA_VALUE, value, sizeof(value) }
2112 } ;

```

## 2.4.4 Diffie-Hellman private key objects

2114 Diffie-Hellman private key objects (object class **CKO\_PRIVATE\_KEY**, key type **CKK\_DH**) hold Diffie-  
2115 Hellman private keys. The following table defines the Diffie-Hellman private key object attributes, in  
2116 addition to the common attributes defined for this object class:

2117 Table 55, Diffie-Hellman Private Key Object Attributes

| Attribute                     | Data type   | Meaning                             |
|-------------------------------|-------------|-------------------------------------|
| CKA_PRIME <sup>1,4,6</sup>    | Big integer | Prime $p$                           |
| CKA_BASE <sup>1,4,6</sup>     | Big integer | Base $g$                            |
| CKA_VALUE <sup>1,4,6,7</sup>  | Big integer | Private value $x$                   |
| CKA_VALUE_BITS <sup>2,6</sup> | CK ULONG    | Length in bits of private value $x$ |

2118 - Refer to [PKCS11-Base] table 11 for footnotes

2119 The **CKA\_PRIME** and **CKA\_BASE** attribute values are collectively the “Diffie-Hellman domain  
 2120 parameters”. Depending on the token, there may be limits on the length of the key components. See  
 2121 PKCS #3 for more information on Diffie-Hellman keys.

2122 Note that when generating a Diffie-Hellman private key, the Diffie-Hellman parameters are *not* specified in  
 2123 the key’s template. This is because Diffie-Hellman private keys are only generated as part of a Diffie-  
 2124 Hellman key pair, and the Diffie-Hellman parameters for the pair are specified in the template for the  
 2125 Diffie-Hellman public key.

2126 The following is a sample template for creating a Diffie-Hellman private key object:

```
2127 CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
2128 CK_KEY_TYPE keyType = CKK_DH;
2129 CK_UTF8CHAR label[] = "A Diffie-Hellman private key object";
2130 CK_BYTE subject[] = {...};
2131 CK_BYTE id[] = {123};
2132 CK_BYTE prime[] = {...};
2133 CK_BYTE base[] = {...};
2134 CK_BYTE value[] = {...};
2135 CK_BBOOL true = CK_TRUE;
2136 CK_ATTRIBUTE template[] = {
2137 {CKA_CLASS, &class, sizeof(class)},
2138 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
2139 {CKA_TOKEN, &true, sizeof(true)},
2140 {CKA_LABEL, label, sizeof(label)-1},
2141 {CKA_SUBJECT, subject, sizeof(subject)},
2142 {CKA_ID, id, sizeof(id)},
2143 {CKA_SENSITIVE, &true, sizeof(true)},
2144 {CKA_DERIVE, &true, sizeof(true)},
2145 {CKA_PRIME, prime, sizeof(prime)},
2146 {CKA_BASE, base, sizeof(base)},
2147 {CKA_VALUE, value, sizeof(value)}}
2148 };
```

## 2149 2.4.5 X9.42 Diffie-Hellman private key objects

2150 X9.42 Diffie-Hellman private key objects (object class **CKO\_PRIVATE\_KEY**, key type **CKK\_X9\_42\_DH**)  
 2151 hold X9.42 Diffie-Hellman private keys. The following table defines the X9.42 Diffie-Hellman private key  
 2152 object attributes, in addition to the common attributes defined for this object class:

2153 Table 56, X9.42 Diffie-Hellman Private Key Object Attributes

| Attribute                     | Data type   | Meaning                                             |
|-------------------------------|-------------|-----------------------------------------------------|
| CKA_PRIME <sup>1,4,6</sup>    | Big integer | Prime $p$ ( $\geq 1024$ bits, in steps of 256 bits) |
| CKA_BASE <sup>1,4,6</sup>     | Big integer | Base $g$                                            |
| CKA_SUBPRIME <sup>1,4,6</sup> | Big integer | Subprime $q$ ( $\geq 160$ bits)                     |
| CKA_VALUE <sup>1,4,6,7</sup>  | Big integer | Private value $x$                                   |

2154 - Refer to [PKCS11-Base] table 11 for footnotes

2155 The **CKA\_PRIME**, **CKA\_BASE** and **CKA\_SUBPRIME** attribute values are collectively the “X9.42 Diffie-  
2156 Hellman domain parameters”. Depending on the token, there may be limits on the length of the key  
2157 components. See the ANSI X9.42 standard for more information on X9.42 Diffie-Hellman keys.2158 Note that when generating a X9.42 Diffie-Hellman private key, the X9.42 Diffie-Hellman domain  
2159 parameters are *not* specified in the key’s template. This is because X9.42 Diffie-Hellman private keys are  
2160 only generated as part of a X9.42 Diffie-Hellman key *pair*, and the X9.42 Diffie-Hellman domain  
2161 parameters for the pair are specified in the template for the X9.42 Diffie-Hellman public key.

2162 The following is a sample template for creating a X9.42 Diffie-Hellman private key object:

```

2163 CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
2164 CK_KEY_TYPE keyType = CKK_X9_42_DH;
2165 CK_UTF8CHAR label[] = "A X9.42 Diffie-Hellman private key object";
2166 CK_BYTE subject[] = {...};
2167 CK_BYTE id[] = {123};
2168 CK_BYTE prime[] = {...};
2169 CK_BYTE base[] = {...};
2170 CK_BYTE subprime[] = {...};
2171 CK_BYTE value[] = {...};
2172 CK_BBOOL true = CK_TRUE;
2173 CK_ATTRIBUTE template[] = {
2174 {CKA_CLASS, &class, sizeof(class)},
2175 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
2176 {CKA_TOKEN, &true, sizeof(true)},
2177 {CKA_LABEL, label, sizeof(label)-1},
2178 {CKA_SUBJECT, subject, sizeof(subject)},
2179 {CKA_ID, id, sizeof(id)},
2180 {CKA_SENSITIVE, &true, sizeof(true)},
2181 {CKA_DERIVE, &true, sizeof(true)},
2182 {CKA_PRIME, prime, sizeof(prime)},
2183 {CKA_BASE, base, sizeof(base)},
2184 {CKA_SUBPRIME, subprime, sizeof(subprime)},
2185 {CKA_VALUE, value, sizeof(value)}
2186 };

```

2187 

## 2.4.6 Diffie-Hellman domain parameter objects

2188 Diffie-Hellman domain parameter objects (object class **CKO\_DOMAIN\_PARAMETERS**, key type  
2189 **CKK\_DH**) hold Diffie-Hellman domain parameters. The following table defines the Diffie-Hellman domain  
2190 parameter object attributes, in addition to the common attributes defined for this object class:

2191 Table 57, Diffie-Hellman Domain Parameter Object Attributes

| Attribute                     | Data type   | Meaning                    |
|-------------------------------|-------------|----------------------------|
| CKA_PRIME <sup>1,4</sup>      | Big integer | Prime $p$                  |
| CKA_BASE <sup>1,4</sup>       | Big integer | Base $g$                   |
| CKA_PRIME_BITS <sup>2,3</sup> | CK ULONG    | Length of the prime value. |

2192 - Refer to [PKCS11-Base] table 11 for footnotes

2193 The **CKA\_PRIME** and **CKA\_BASE** attribute values are collectively the “Diffie-Hellman domain  
 2194 parameters”. Depending on the token, there may be limits on the length of the key components. See  
 2195 PKCS #3 for more information on Diffie-Hellman domain parameters.

2196 The following is a sample template for creating a Diffie-Hellman domain parameter object:

```

2197 CK_OBJECT_CLASS class = CKO_DOMAIN_PARAMETERS;
2198 CK_KEY_TYPE keyType = CKK_DH;
2199 CK_UTF8CHAR label[] = "A Diffie-Hellman domain parameters
2200 object";
2201 CK_BYTE prime[] = {...};
2202 CK_BYTE base[] = {...};
2203 CK_BBOOL true = CK_TRUE;
2204 CK_ATTRIBUTE template[] = {
2205 {CKA_CLASS, &class, sizeof(class)},
2206 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
2207 {CKA_TOKEN, &true, sizeof(true)},
2208 {CKA_LABEL, label, sizeof(label)-1},
2209 {CKA_PRIME, prime, sizeof(prime)},
2210 {CKA_BASE, base, sizeof(base)}},
2211 };

```

## 2212 2.4.7 X9.42 Diffie-Hellman domain parameters objects

2213 X9.42 Diffie-Hellman domain parameters objects (object class **CKO\_DOMAIN\_PARAMETERS**, key type  
 2214 **CKK\_X9\_42\_DH**) hold X9.42 Diffie-Hellman domain parameters. The following table defines the X9.42  
 2215 Diffie-Hellman domain parameters object attributes, in addition to the common attributes defined for this  
 2216 object class:

2217 Table 58, X9.42 Diffie-Hellman Domain Parameters Object Attributes

| Attribute                        | Data type   | Meaning                                             |
|----------------------------------|-------------|-----------------------------------------------------|
| CKA_PRIME <sup>1,4</sup>         | Big integer | Prime $p$ ( $\geq 1024$ bits, in steps of 256 bits) |
| CKA_BASE <sup>1,4</sup>          | Big integer | Base $g$                                            |
| CKA_SUBPRIME <sup>1,4</sup>      | Big integer | Subprime $q$ ( $\geq 160$ bits)                     |
| CKA_PRIME_BITS <sup>2,3</sup>    | CK ULONG    | Length of the prime value.                          |
| CKA_SUBPRIME_BITS <sup>2,3</sup> | CK ULONG    | Length of the subprime value.                       |

2218 - Refer to [PKCS11-Base] table 11 for footnotes

2219 The **CKA\_PRIME**, **CKA\_BASE** and **CKA\_SUBPRIME** attribute values are collectively the “X9.42 Diffie-  
 2220 Hellman domain parameters”. Depending on the token, there may be limits on the length of the domain  
 2221 parameters components. See the ANSI X9.42 standard for more information on X9.42 Diffie-Hellman  
 2222 domain parameters.

2223 The following is a sample template for creating a X9.42 Diffie-Hellman domain parameters object:

```

2224 CK_OBJECT_CLASS class = CKO_DOMAIN_PARAMETERS;
2225 CK_KEY_TYPE keyType = CKK_X9_42_DH;

```

```

2226 CK_UTF8CHAR label[] = "A X9.42 Diffie-Hellman domain
2227 parameters object";
2228 CK_BYTE prime[] = {...};
2229 CK_BYTE base[] = {...};
2230 CK_BYTE subprime[] = {...};
2231 CK_BBOOL true = CK_TRUE;
2232 CK_ATTRIBUTE template[] = {
2233 {CKA_CLASS, &class, sizeof(class)},
2234 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
2235 {CKA_TOKEN, &true, sizeof(true)},
2236 {CKA_LABEL, label, sizeof(label)-1},
2237 {CKA_PRIME, prime, sizeof(prime)},
2238 {CKA_BASE, base, sizeof(base)},
2239 {CKA_SUBPRIME, subprime, sizeof(subprime)},
2240 };

```

#### 2241 **2.4.8 PKCS #3 Diffie-Hellman key pair generation**

2242 The PKCS #3 Diffie-Hellman key pair generation mechanism, denoted  
2243 **CKM\_DH\_PKCS\_KEY\_PAIR\_GEN**, is a key pair generation mechanism based on Diffie-Hellman key  
2244 agreement, as defined in PKCS #3. This is what PKCS #3 calls “phase I”. It does not have a parameter.  
2245 The mechanism generates Diffie-Hellman public/private key pairs with a particular prime and base, as  
2246 specified in the **CKA\_PRIME** and **CKA\_BASE** attributes of the template for the public key. If the  
2247 **CKA\_VALUE\_BITS** attribute of the private key is specified, the mechanism limits the length in bits of the  
2248 private value, as described in PKCS #3.  
2249 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
2250 public key and the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_PRIME**, **CKA\_BASE**, and **CKA\_VALUE** (and  
2251 the **CKA\_VALUE\_BITS** attribute, if it is not already provided in the template) attributes to the new private  
2252 key; other attributes required by the Diffie-Hellman public and private key types must be specified in the  
2253 templates.  
2254 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2255 specify the supported range of Diffie-Hellman prime sizes, in bits.

#### 2256 **2.4.9 PKCS #3 Diffie-Hellman domain parameter generation**

2257 The PKCS #3 Diffie-Hellman domain parameter generation mechanism, denoted  
2258 **CKM\_DH\_PKCS\_PARAMETER\_GEN**, is a domain parameter generation mechanism based on Diffie-  
2259 Hellman key agreement, as defined in PKCS #3.  
2260 It does not have a parameter.  
2261 The mechanism generates Diffie-Hellman domain parameters with a particular prime length in bits, as  
2262 specified in the **CKA\_PRIME\_BITS** attribute of the template.  
2263 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_PRIME**, **CKA\_BASE**, and  
2264 **CKA\_PRIME\_BITS** attributes to the new object. Other attributes supported by the Diffie-Hellman domain  
2265 parameter types may also be specified in the template, or else are assigned default initial values.  
2266 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2267 specify the supported range of Diffie-Hellman prime sizes, in bits.

#### 2268 **2.4.10 PKCS #3 Diffie-Hellman key derivation**

2269 The PKCS #3 Diffie-Hellman key derivation mechanism, denoted **CKM\_DH\_PKCS\_DERIVE**, is a  
2270 mechanism for key derivation based on Diffie-Hellman key agreement, as defined in PKCS #3. This is  
2271 what PKCS #3 calls “phase II”.

2272 It has a parameter, which is the public value of the other party in the key agreement protocol, represented  
2273 as a Cryptoki “Big integer” (i.e., a sequence of bytes, most-significant byte first).

2274 This mechanism derives a secret key from a Diffie-Hellman private key and the public value of the other  
2275 party. It computes a Diffie-Hellman secret value from the public value and private key according to PKCS  
2276 #3, and truncates the result according to the **CKA\_KEY\_TYPE** attribute of the template and, if it has one  
2277 and the key type supports it, the **CKA\_VALUE\_LEN** attribute of the template. (The truncation removes  
2278 bytes from the leading end of the secret value.) The mechanism contributes the result as the  
2279 **CKA\_VALUE** attribute of the new key; other attributes required by the key type must be specified in the  
2280 template.

2281 This mechanism has the following rules about key sensitivity and extractability<sup>2</sup>:

- 2282 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
2283 be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
2284 default value.
- 2285 • If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
2286 will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the  
2287 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
2288 **CKA\_SENSITIVE** attribute.
- 2289 • Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
2290 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
2291 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
2292 value from its **CKA\_EXTRACTABLE** attribute.

2293 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2294 specify the supported range of Diffie-Hellman prime sizes, in bits.

#### 2295 2.4.11 X9.42 Diffie-Hellman mechanism parameters

2296 ♦ **CK\_X9\_42\_DH\_KDF\_TYPE**, **CK\_X9\_42\_DH\_KDF\_TYPE\_PTR**

2297 **CK\_X9\_42\_DH\_KDF\_TYPE** is used to indicate the Key Derivation Function (KDF) applied to derive  
2298 keying data from a shared secret. The key derivation function will be used by the X9.42 Diffie-Hellman  
2299 key agreement schemes. It is defined as follows:

2300       **typedef CK ULONG CK\_X9\_42\_DH\_KDF\_TYPE;**  
2301

2302 The following table lists the defined functions.

2303 *Table 59, X9.42 Diffie-Hellman Key Derivation Functions*

| Source Identifier        |
|--------------------------|
| CKD_NULL                 |
| CKD_SHA1_KDF ASN1        |
| CKD_SHA1_KDF CONCATENATE |

2304 The key derivation function **CKD\_NULL** produces a raw shared secret value without applying any key  
2305 derivation function whereas the key derivation functions **CKD\_SHA1\_KDF ASN1** and  
2306 **CKD\_SHA1\_KDF CONCATENATE**, which are both based on SHA-1, derive keying data from the  
2307 shared secret value as defined in the ANSI X9.42 standard.

2308 **CK\_X9\_42\_DH\_KDF\_TYPE\_PTR** is a pointer to a **CK\_X9\_42\_DH\_KDF\_TYPE**.

---

2 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.

2309 ◆ **CK\_X9\_42\_DH1\_DERIVE\_PARAMS, CK\_X9\_42\_DH1\_DERIVE\_PARAMS\_PTR**

2310 **CK\_X9\_42\_DH1\_DERIVE\_PARAMS** is a structure that provides the parameters to the  
2311 **CKM\_X9\_42\_DH\_DERIVE** key derivation mechanism, where each party contributes one key pair. The  
2312 structure is defined as follows:

```
2313 typedef struct CK_X9_42_DH1_DERIVE_PARAMS {
2314 CK_X9_42_DH_KDF_TYPE kdf;
2315 CK ULONG ulOtherInfoLen;
2316 CK BYTE PTR pOtherInfo;
2317 CK ULONG ulPublicDataLen;
2318 CK BYTE PTR pPublicData;
2319 } CK_X9_42_DH1_DERIVE_PARAMS;
```

2320 The fields of the structure have the following meanings:

2322                    *kdf*      key derivation function used on the shared secret value

2323                    *ulOtherInfoLen*     the length in bytes of the other info

2324                    *pOtherInfo*     some data shared between the two parties

2325                    *ulPublicDataLen*     the length in bytes of the other party's X9.42 Diffie-Hellman public  
2326                    key

2327                    *pPublicData*     pointer to other party's X9.42 Diffie-Hellman public key value

2328 With the key derivation function **CKD\_NULL**, *pOtherInfo* must be NULL and *ulOtherInfoLen* must be zero.  
2329 With the key derivation function **CKD\_SHA1\_KDF ASN1**, *pOtherInfo* must be supplied, which contains  
2330 an octet string, specified in ASN.1 DER encoding, consisting of mandatory and optional data shared by  
2331 the two parties intending to share the shared secret. With the key derivation function  
2332 **CKD\_SHA1\_KDF CONCATENATE**, an optional *pOtherInfo* may be supplied, which consists of some  
2333 data shared by the two parties intending to share the shared secret. Otherwise, *pOtherInfo* must be  
2334 NULL and *ulOtherInfoLen* must be zero.

2335 **CK\_X9\_42\_DH1\_DERIVE\_PARAMS\_PTR** is a pointer to a **CK\_X9\_42\_DH1\_DERIVE\_PARAMS**.

2336 • **CK\_X9\_42\_DH2\_DERIVE\_PARAMS, CK\_X9\_42\_DH2\_DERIVE\_PARAMS\_PTR**

2337 **CK\_X9\_42\_DH2\_DERIVE\_PARAMS** is a structure that provides the parameters to the  
2338 **CKM\_X9\_42\_DH\_HYBRID\_DERIVE** and **CKM\_X9\_42\_MQV\_DERIVE** key derivation mechanisms,  
2339 where each party contributes two key pairs. The structure is defined as follows:

```
2340 typedef struct CK_X9_42_DH2_DERIVE_PARAMS {
2341 CK_X9_42_DH_KDF_TYPE kdf;
2342 CK ULONG ulOtherInfoLen;
2343 CK BYTE PTR pOtherInfo;
2344 CK ULONG ulPublicDataLen;
2345 CK BYTE PTR pPublicData;
2346 CK ULONG ulPrivateDataLen;
2347 CK OBJECT_HANDLE hPrivateData;
2348 CK ULONG ulPublicDataLen2;
2349 CK BYTE PTR pPublicData2;
2350 } CK_X9_42_DH2_DERIVE_PARAMS;
```

2351  
2352     The fields of the structure have the following meanings:  
2353                 *kdf*     *key derivation function used on the shared secret value*  
2354                 *ulOtherInfoLen*     *the length in bytes of the other info*  
2355                 *pOtherInfo*     *some data shared between the two parties*  
2356                 *ulPublicDataLen*     *the length in bytes of the other party's first X9.42 Diffie-Hellman public key*  
2357  
2358                 *pPublicData*     *pointer to other party's first X9.42 Diffie-Hellman public key value*  
2359                 *ulPrivateDataLen*     *the length in bytes of the second X9.42 Diffie-Hellman private key*  
2360                 *hPrivateKey*     *key handle for second X9.42 Diffie-Hellman private key value*  
2361                 *ulPublicDataLen2*     *the length in bytes of the other party's second X9.42 Diffie-Hellman public key*  
2362  
2363                 *pPublicData2*     *pointer to other party's second X9.42 Diffie-Hellman public key value*  
2364

2365     With the key derivation function **CKD\_NULL**, *pOtherInfo* must be NULL and *ulOtherInfoLen* must be zero.  
2366     With the key derivation function **CKD\_SHA1\_KDF ASN1**, *pOtherInfo* must be supplied, which contains  
2367     an octet string, specified in ASN.1 DER encoding, consisting of mandatory and optional data shared by  
2368     the two parties intending to share the shared secret. With the key derivation function  
2369     **CKD\_SHA1\_KDF\_CONCATENATE**, an optional *pOtherInfo* may be supplied, which consists of some  
2370     data shared by the two parties intending to share the shared secret. Otherwise, *pOtherInfo* must be  
2371     NULL and *ulOtherInfoLen* must be zero.

2372     **CK\_X9\_42\_DH2\_DERIVE\_PARAMS\_PTR** is a pointer to a **CK\_X9\_42\_DH2\_DERIVE\_PARAMS**.

2373     • **CK\_X9\_42\_MQV\_DERIVE\_PARAMS, CK\_X9\_42\_MQV\_DERIVE\_PARAMS\_PTR**

2374     **CK\_X9\_42\_MQV\_DERIVE\_PARAMS** is a structure that provides the parameters to the  
2375     **CKM\_X9\_42\_MQV\_DERIVE** key derivation mechanism, where each party contributes two key pairs. The  
2376     structure is defined as follows:

```
2377 typedef struct CK_X9_42_MQV_DERIVE_PARAMS {
2378 CK_X9_42_DH_KDF_TYPE kdf;
2379 CK ULONG ulOtherInfoLen;
2380 CK_BYTE_PTR pOtherInfo;
2381 CK ULONG ulPublicDataLen;
2382 CK_BYTE_PTR pPublicData;
2383 CK ULONG ulPrivateDataLen;
2384 CK_OBJECT_HANDLE hPrivateKey;
2385 CK ULONG ulPublicDataLen2;
2386 CK_BYTE_PTR pPublicData2;
2387 CK_OBJECT_HANDLE publicKey;
2388 } CK_X9_42_MQV_DERIVE_PARAMS;
```

2389  
2390     The fields of the structure have the following meanings:  
2391                 *kdf*     *key derivation function used on the shared secret value*  
2392                 *ulOtherInfoLen*     *the length in bytes of the other info*  
2393                 *pOtherInfo*     *some data shared between the two parties*  
2394                 *ulPublicDataLen*     *the length in bytes of the other party's first X9.42 Diffie-Hellman public key*  
2395  
2396                 *pPublicData*     *pointer to other party's first X9.42 Diffie-Hellman public key value*  
2397                 *ulPrivateDataLen*     *the length in bytes of the second X9.42 Diffie-Hellman private key*  
2398                 *hPrivateData*     *key handle for second X9.42 Diffie-Hellman private key value*  
2399                 *ulPublicDataLen2*     *the length in bytes of the other party's second X9.42 Diffie-Hellman public key*  
2400  
2401                 *pPublicData2*     *pointer to other party's second X9.42 Diffie-Hellman public key value*  
2402  
2403                 *publicKey*     *Handle to the first party's ephemeral public key*

2404     With the key derivation function **CKD\_NULL**, *pOtherInfo* must be NULL and *ulOtherInfoLen* must be zero.  
2405     With the key derivation function **CKD\_SHA1\_KDF ASN1**, *pOtherInfo* must be supplied, which contains  
2406     an octet string, specified in ASN.1 DER encoding, consisting of mandatory and optional data shared by  
2407     the two parties intending to share the shared secret. With the key derivation function  
2408     **CKD\_SHA1\_KDF CONCATENATE**, an optional *pOtherInfo* may be supplied, which consists of some  
2409     data shared by the two parties intending to share the shared secret. Otherwise, *pOtherInfo* must be  
2410     NULL and *ulOtherInfoLen* must be zero.

2411     **CK\_X9\_42\_MQV\_DERIVE\_PARAMS\_PTR** is a pointer to a **CK\_X9\_42\_MQV\_DERIVE\_PARAMS**.

## 2412     2.4.12 X9.42 Diffie-Hellman key pair generation

2413     The X9.42 Diffie-Hellman key pair generation mechanism, denoted **CKM\_X9\_42\_DH\_KEY\_PAIR\_GEN**,  
2414     is a key pair generation mechanism based on Diffie-Hellman key agreement, as defined in the ANSI  
2415     X9.42 standard.

2416     It does not have a parameter.

2417     The mechanism generates X9.42 Diffie-Hellman public/private key pairs with a particular prime, base and  
2418     subprime, as specified in the **CKA\_PRIME**, **CKA\_BASE** and **CKA\_SUBPRIME** attributes of the template  
2419     for the public key.

2420     The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
2421     public key and the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_PRIME**, **CKA\_BASE**, **CKA\_SUBPRIME**, and  
2422     **CKA\_VALUE** attributes to the new private key; other attributes required by the X9.42 Diffie-Hellman  
2423     public and private key types must be specified in the templates.

2424     For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2425     specify the supported range of X9.42 Diffie-Hellman prime sizes, in bits, for the **CKA\_PRIME** attribute.

2426 **2.4.13 X9.42 Diffie-Hellman domain parameter generation**

2427 The X9.42 Diffie-Hellman domain parameter generation mechanism, denoted  
2428 **CKM\_X9\_42\_DH\_PARAMETER\_GEN**, is a domain parameters generation mechanism based on X9.42  
2429 Diffie-Hellman key agreement, as defined in the ANSI X9.42 standard.

2430 It does not have a parameter.

2431 The mechanism generates X9.42 Diffie-Hellman domain parameters with particular prime and subprime  
2432 length in bits, as specified in the **CKA\_PRIME\_BITS** and **CKA\_SUBPRIME\_BITS** attributes of the  
2433 template for the domain parameters.

2434 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_PRIME**, **CKA\_BASE**,  
2435 **CKA\_SUBPRIME**, **CKA\_PRIME\_BITS** and **CKA\_SUBPRIME\_BITS** attributes to the new object. Other  
2436 attributes supported by the X9.42 Diffie-Hellman domain parameter types may also be specified in the  
2437 template for the domain parameters, or else are assigned default initial values.

2438 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2439 specify the supported range of X9.42 Diffie-Hellman prime sizes, in bits.

2440 **2.4.14 X9.42 Diffie-Hellman key derivation**

2441 The X9.42 Diffie-Hellman key derivation mechanism, denoted **CKM\_X9\_42\_DH\_DERIVE**, is a  
2442 mechanism for key derivation based on the Diffie-Hellman key agreement scheme, as defined in the  
2443 ANSI X9.42 standard, where each party contributes one key pair, all using the same X9.42 Diffie-Hellman  
2444 domain parameters.

2445 It has a parameter, a **CK\_X9\_42\_DH1\_DERIVE\_PARAMS** structure.

2446 This mechanism derives a secret value, and truncates the result according to the **CKA\_KEY\_TYPE**  
2447 attribute of the template and, if it has one and the key type supports it, the **CKA\_VALUE\_LEN** attribute of  
2448 the template. (The truncation removes bytes from the leading end of the secret value.) The mechanism  
2449 contributes the result as the **CKA\_VALUE** attribute of the new key; other attributes required by the key  
2450 type must be specified in the template. Note that in order to validate this mechanism it may be required to  
2451 use the **CKA\_VALUE** attribute as the key of a general-length MAC mechanism (e.g.  
2452 **CKM\_SHA\_1\_HMAC\_GENERAL**) over some test data.

2453 This mechanism has the following rules about key sensitivity and extractability:

- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
2455 be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
2456 default value.
- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
2458 will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the  
2459 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
2460 **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
2462 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
2463 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
2464 value from its **CKA\_EXTRACTABLE** attribute.

2465 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2466 specify the supported range of X9.42 Diffie-Hellman prime sizes, in bits, for the **CKA\_PRIME** attribute.

2467 **2.4.15 X9.42 Diffie-Hellman hybrid key derivation**

2468 The X9.42 Diffie-Hellman hybrid key derivation mechanism, denoted  
2469 **CKM\_X9\_42\_DH\_HYBRID\_DERIVE**, is a mechanism for key derivation based on the Diffie-Hellman  
2470 hybrid key agreement scheme, as defined in the ANSI X9.42 standard, where each party contributes two  
2471 key pair, all using the same X9.42 Diffie-Hellman domain parameters.

2472 It has a parameter, a **CK\_X9\_42\_DH2\_DERIVE\_PARAMS** structure.

2473 This mechanism derives a secret value, and truncates the result according to the **CKA\_KEY\_TYPE**  
2474 attribute of the template and, if it has one and the key type supports it, the **CKA\_VALUE\_LEN** attribute of  
2475 the template. (The truncation removes bytes from the leading end of the secret value.) The mechanism  
2476 contributes the result as the **CKA\_VALUE** attribute of the new key; other attributes required by the key  
2477 type must be specified in the template. Note that in order to validate this mechanism it may be required to  
2478 use the **CKA\_VALUE** attribute as the key of a general-length MAC mechanism (e.g.  
2479 **CKM\_SHA\_1\_HMAC\_GENERAL**) over some test data.

2480 This mechanism has the following rules about key sensitivity and extractability:

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

2482 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2483 specify the supported range of X9.42 Diffie-Hellman prime sizes, in bits, for the **CKA\_PRIME** attribute.

#### 2494 2.4.16 X9.42 Diffie-Hellman Menezes-Qu-Vanstone key derivation

2495 The X9.42 Diffie-Hellman Menezes-Qu-Vanstone (MQV) key derivation mechanism, denoted  
2496 **CKM\_X9\_42\_MQV\_DERIVE**, is a mechanism for key derivation based the MQV scheme, as defined in  
2497 the ANSI X9.42 standard, where each party contributes two key pairs, all using the same X9.42 Diffie-  
2498 Hellman domain parameters.

2499 It has a parameter, a **CK\_X9\_42\_MQV\_DERIVE\_PARAMS** structure.

2500 This mechanism derives a secret value, and truncates the result according to the **CKA\_KEY\_TYPE**  
2501 attribute of the template and, if it has one and the key type supports it, the **CKA\_VALUE\_LEN** attribute of  
2502 the template. (The truncation removes bytes from the leading end of the secret value.) The mechanism  
2503 contributes the result as the **CKA\_VALUE** attribute of the new key; other attributes required by the key  
2504 type must be specified in the template. Note that in order to validate this mechanism it may be required to  
2505 use the **CKA\_VALUE** attribute as the key of a general-length MAC mechanism (e.g.  
2506 **CKM\_SHA\_1\_HMAC\_GENERAL**) over some test data.

2507 This mechanism has the following rules about key sensitivity and extractability:

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

2519 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2520 specify the supported range of X9.42 Diffie-Hellman prime sizes, in bits, for the **CKA\_PRIME** attribute.

## 2521 2.5 Extended Triple Diffie-Hellman (x3dh)

2522 The Extended Triple Diffie-Hellman mechanism described here is the one described in  
2523 [SIGNAL].

2524

2525 *Table 60, Extended Triple Diffie-Hellman Mechanisms vs. Functions*

| Mechanism           | Functions         |               |                      |        |                    |               |        |
|---------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                     | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_X3DH_INITIALIZE |                   |               |                      |        |                    |               | ✓      |
| CKM_X3DH RESPOND    |                   |               |                      |        |                    |               | ✓      |

### 2526 2.5.1 Definitions

2527 Mechanisms:

2528 CKM\_X3DH\_INITIALIZE

2529 CKM\_X3DH\_RESPOND

### 2530 2.5.2 Extended Triple Diffie-Hellman key objects

2531 Extended Triple Diffie-Hellman uses Elliptic Curve keys in Montgomery representation

2532 (**CKK\_EC\_MONTGOMERY**). Three different kinds of keys are used, they differ in their lifespan:

- 2533 • identity keys are long-term keys, which identify the peer,
- 2534 • prekeys are short-term keys, which should be rotated often (weekly to hourly)
- 2535 • onetime prekeys are keys, which should be used only once.

2536 Any peer intending to be contacted using X3DH must publish their so-called prekey-bundle, consisting of  
2537 their:

- 2538 • public Identity key,
- 2539 • current prekey, signed using XEDDA with their identity key
- 2540 • optionally a batch of One-time public keys.

### 2541 2.5.3 Initiating an Extended Triple Diffie-Hellman key exchange

2542 Initiating an Extended Triple Diffie-Hellman key exchange starts by retrieving the following required public  
2543 keys (the so-called prekey-bundle) of the other peer: the Identity key, the signed public Prekey, and  
2544 optionally one One-time public key.

2545 When the necessary key material is available, the initiating party calls CKM\_X3DH\_INITIALIZE, also  
2546 providing the following additional parameters:

- 2547 • the initiators identity key
- 2548 • the initiators ephemeral key (a fresh, one-time **CKK\_EC\_MONTGOMERY** type key)

2549  
2550 **CK\_X3DH\_INITIATE\_PARAMS** is a structure that provides the parameters to the  
2551 **CKM\_X3DH\_INITIALIZE** key exchange mechanism. The structure is defined as follows:

```
2552 typedef struct CK_X3DH_INITIATE_PARAMS {
2553 CK_X3DH_KDF_TYPE kdf;
2554 CK_OBJECT_HANDLE pPeer_identity;
```

```

2555 CK_OBJECT_HANDLE pPeer_prekey;
2556 CK_BYTE_PTR pPrekey_signature;
2557 CK_BYTE_PTR pOnetime_key;
2558 CK_OBJECT_HANDLE pOwn_identity;
2559 CK_OBJECT_HANDLE pOwn_ephemeral;
2560 } CK_X3DH_INITIATE_PARAMS;

```

2561 *Table 61, Extended Triple Diffie-Hellman Initiate Message parameters:*

| <b>Parameter</b>  | <b>Data type</b> | <b>Meaning</b>                                                   |
|-------------------|------------------|------------------------------------------------------------------|
| kdf               | CK_X3DH_KDF_TYPE | <i>Key derivation function</i>                                   |
| pPeer_identity    | Key handle       | <i>Peers public Identity key (from the prekey-bundle)</i>        |
| pPeer_prekey      | Key Handle       | <i>Peers public prekey (from the prekey-bundle)</i>              |
| pPrekey_signature | Byte array       | <i>XEDDSA signature of PEER_PREKEY (from prekey-bundle)</i>      |
| pOnetime_key      | Byte array       | Optional one-time public prekey of peer (from the prekey-bundle) |
| pOwn_identity     | Key Handle       | Initiators Identity key                                          |
| pOwn_ephemeral    | Key Handle       | Initiators ephemeral key                                         |

2562

## 2563 **2.5.4 Responding to an Extended Triple Diffie-Hellman key exchange**

2564 Responding an Extended Triple Diffie-Hellman key exchange is done by executing a  
2565 CKM\_X3DH\_RESPOND mechanism. **CK\_X3DH\_RESPOND\_PARAMS** is a structure that provides the  
2566 parameters to the **CKM\_X3DH\_RESPOND** key exchange mechanism. All these parameter should be  
2567 supplied by the Initiator in a message to the responder. The structure is defined as follows:

```

2568 typedef struct CK_X3DH_RESPOND_PARAMS {
2569 CK_X3DH_KDF_TYPE kdf;
2570 CK_BYTE_PTR pIdentity_id;
2571 CK_BYTE_PTR pPrekey_id;
2572 CK_BYTE_PTR pOnetime_id;
2573 CK_OBJECT_HANDLE pInitiator_identity;
2574 CK_BYTE_PTR pInitiator_ephemeral;
2575 } CK_X3DH_RESPOND_PARAMS;

```

2576

2577 *Table 62, Extended Triple Diffie-Hellman 1st Message parameters:*

| <b>Parameter</b>     | <b>Data type</b> | <b>Meaning</b>                                                       |
|----------------------|------------------|----------------------------------------------------------------------|
| kdf                  | CK_X3DH_KDF_TYPE | <i>Key derivation function</i>                                       |
| pIdentity_id         | Byte array       | <i>Peers public Identity key identifier (from the prekey-bundle)</i> |
| pPrekey_id           | Byte array       | <i>Peers public prekey identifier (from the prekey-bundle)</i>       |
| pOnetime_id          | Byte array       | Optional one-time public prekey of peer (from the prekey-bundle)     |
| pInitiator_identity  | Key handle       | Initiators Identity key                                              |
| pInitiator_ephemeral | Byte array       | Initiators ephemeral key                                             |

2578

2579 Where the \*\_id fields are identifiers marking which key has been used from the prekey-bundle, these  
2580 identifiers could be the keys themselves.

2581

2582 This mechanism has the following rules about key sensitivity and extractability<sup>3</sup>:

- 2583 1 The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
2584 be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
2585 default value.
- 2586 2 If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
2587 will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the  
2588 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
2589 **CKA\_SENSITIVE** attribute.
- 2590 3 Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
2591 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
2592 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
2593 value from its **CKA\_EXTRACTABLE** attribute.

2594 **2.5.5 Extended Triple Diffie-Hellman parameters**

2595 • **CK\_X3DH\_KDF\_TYPE, CK\_X3DH\_KDF\_TYPE\_PTR**

2596 **CK\_X3DH\_KDF\_TYPE** is used to indicate the Key Derivation Function (KDF) applied to derive keying  
2597 data from a shared secret. The key derivation function will be used by the X3DH key agreement  
2598 schemes. It is defined as follows:

2599     **typedef CK ULONG CK\_X3DH\_KDF\_TYPE;**

2600

2601 The following table lists the defined functions.

2602 *Table 63, X3DH: Key Derivation Functions*

| Source Identifier   |
|---------------------|
| CKD_NULL            |
| CKD_BLAKE2B_256_KDF |
| CKD_BLAKE2B_512_KDF |
| CKD_SHA3_256_KDF    |
| CKD_SHA256_KDF      |
| CKD_SHA3_512_KDF    |
| CKD_SHA512_KDF      |

2603 **2.6 Double Ratchet**

2604 The Double Ratchet is a key management algorithm managing the ongoing renewal and maintenance of  
2605 short-lived session keys providing forward secrecy and break-in recovery for encrypt/decrypt operations.  
2606 The algorithm is described in [DoubleRatchet]. The Signal protocol uses X3DH to exchange a shared  
2607 secret in the first step, which is then used to derive a Double Ratchet secret key.

---

3 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 64, Double Ratchet Mechanisms vs. Functions

| <b>Mechanism</b>         | <b>Functions</b>             |                          |                                    |               |                               |                          |               |
|--------------------------|------------------------------|--------------------------|------------------------------------|---------------|-------------------------------|--------------------------|---------------|
|                          | <b>Encrypt &amp; Decrypt</b> | <b>Sign &amp; Verify</b> | <b>SR &amp;<br/>VR<sup>1</sup></b> | <b>Digest</b> | <b>Gen. Key/<br/>Key Pair</b> | <b>Wrap &amp; Unwrap</b> | <b>Derive</b> |
| CKM_X2RATCHET_INITIALIZE |                              |                          |                                    |               |                               |                          | ✓             |
| CKM_X2RATCHET RESPOND    |                              |                          |                                    |               |                               |                          | ✓             |
| CKM_X2RATCHET_ENCRYPT    | ✓                            |                          |                                    |               |                               | ✓                        |               |
| CKM_X2RATCHET_DECRYPT    | ✓                            |                          |                                    |               |                               | ✓                        |               |

2609

## 2.6.1 Definitions

2611 This section defines the key type "CKK\_X2RATCHET" for type CK\_KEY\_TYPE as used in the  
 2612 CKA\_KEY\_TYPE attribute of key objects.

2613 Mechanisms:

2614 CKM\_X2RATCHET\_INITIALIZE  
 2615 CKM\_X2RATCHET\_RESPOND  
 2616 CKM\_X2RATCHET\_ENCRYPT  
 2617 CKM\_X2RATCHET\_DECRYPT

## 2.6.2 Double Ratchet secret key objects

2619 Double Ratchet secret key objects (object class CKO\_SECRET\_KEY, key type CKK\_X2RATCHET) hold  
 2620 Double Ratchet keys. Double Ratchet secret keys can only be derived from shared secret keys using the  
 2621 mechanism CKM\_X2RATCHET\_INITIALIZE or CKM\_X2RATCHET\_RESPOND. In the Signal protocol  
 2622 these are seeded with the shared secret derived from an Extended Triple Diffie-Hellman [X3DH] key-  
 2623 exchange. The following table defines the Double Ratchet secret key object attributes, in addition to the  
 2624 common attributes defined for this object class:

2625 Table 65, Double Ratchet Secret Key Object Attributes

| <b>Attribute</b>         | <b>Data type</b> | <b>Meaning</b>                              |
|--------------------------|------------------|---------------------------------------------|
| CKA_X2RATCHET_RK         | Byte array       | Root key                                    |
| CKA_X2RATCHET_HKS        | Byte array       | Sender Header key                           |
| CKA_X2RATCHET_HKR        | Byte array       | Receiver Header key                         |
| CKA_X2RATCHET_NHKS       | Byte array       | Next Sender Header Key                      |
| CKA_X2RATCHET_NHKR       | Byte array       | Next Receiver Header Key                    |
| CKA_X2RATCHET_CKS        | Byte array       | Sender Chain key                            |
| CKA_X2RATCHET_CKR        | Byte array       | Receiver Chain key                          |
| CKA_X2RATCHET_DHS        | Byte array       | Sender DH secret key                        |
| CKA_X2RATCHET_DHP        | Byte array       | Sender DH public key                        |
| CKA_X2RATCHET_DHR        | Byte array       | Receiver DH public key                      |
| CKA_X2RATCHET_NS         | ULONG            | Message number send                         |
| CKA_X2RATCHET_NR         | ULONG            | Message number receive                      |
| CKA_X2RATCHET_PNS        | ULONG            | Previous message number send                |
| CKA_X2RATCHET_BOBS1STMSG | BOOL             | Is this bob and has he ever sent a message? |
| CKA_X2RATCHET_ISALICE    | BOOL             | Is this Alice?                              |
| CKA_X2RATCHET_BAGSIZE    | ULONG            | How many out-of-order keys do we store      |

| Attribute         | Data type  | Meaning           |
|-------------------|------------|-------------------|
| CKA_X2RATCHET_BAG | Byte array | Out-of-order keys |

## 2.6.3 Double Ratchet key derivation

The Double Ratchet key derivation mechanisms depend on who is the initiating party, and who the receiving, denoted **CKM\_X2RATCHET\_INITIALIZE** and **CKM\_X2RATCHET\_RESPOND**, are the key derivation mechanisms for the Double Ratchet. Usually the keys are derived from a shared secret by executing a X3DH key exchange.

The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new key. Additionally the attribute flags indicating which functions the key supports are also contributed by the mechanism.

For this mechanism, the only allowed values are 255 and 448 as RFC 8032 only defines curves of these two sizes. A Cryptoki implementation may support one or both of these curves and should set the *ulMinKeySize* and *ulMaxKeySize* fields accordingly.

- **CK\_X2RATCHET\_INITIALIZE\_PARAMS;**  
**CK\_X2RATCHET\_INITIALIZE\_PARAMS\_PTR**

**CK\_X2RATCHET\_INITIALIZE\_PARAMS** provides the parameters to the **CKM\_X2RATCHET\_INITIALIZE** mechanism. It is defined as follows:

```
2639 typedef struct CK_X2RATCHET_INITIALIZE_PARAMS {
2640 CK_BYTE_PTR sk;
2641 CK_OBJECT_HANDLE peer_public_prekey;
2642 CK_OBJECT_HANDLE peer_public_identity;
2643 CK_OBJECT_HANDLE own_public_identity;
2644 CK_BBOOL bEncryptedHeader;
2645 CK ULONG eCurve;
2646 CK_MECHANISM_TYPE aeadMechanism;
2647 CK_X2RATCHET_KDF_TYPE kdfMechanism;
2648 } CK_X2RATCHET_INITIALIZE_PARAMS;
```

The fields of the structure have the following meanings:

|      |                              |                                                                   |
|------|------------------------------|-------------------------------------------------------------------|
| 2653 | <i>sk</i>                    | <i>the shared secret with peer (derived using X3DH)</i>           |
| 2654 | <i>peers_public_prekey</i>   | <i>Peers public prekey which the Initiator used in the X3DH</i>   |
| 2655 | <i>peers_public_identity</i> | <i>Peers public identity which the Initiator used in the X3DH</i> |
| 2656 | <i>own_public_identity</i>   | <i>Initiators public identity as used in the X3DH</i>             |
| 2657 | <i>bEncryptedHeader</i>      | <i>whether the headers are encrypted</i>                          |
| 2658 | <i>eCurve</i>                | <i>255 for curve 25519 or 448 for curve 448</i>                   |
| 2659 | <i>aeadMechanism</i>         | <i>a mechanism supporting AEAD encryption</i>                     |
| 2660 | <i>kdfMechanism</i>          | <i>a Key Derivation Mechanism, such as CKD_BLAKE2B_512_KDF</i>    |

- 2662 • **CK\_X2RATCHET\_RESPOND\_PARAMS;**  
 2663 **CK\_X2RATCHET\_RESPOND\_PARAMS\_PTR**

2664 **CK\_X2RATCHET\_RESPOND\_PARAMS** provides the parameters to the  
 2665 **CKM\_X2RATCHET\_RESPOND** mechanism. It is defined as follows:

```
2666 typedef struct CK_X2RATCHET_RESPOND_PARAMS {
2667 CK_BYTE_PTR sk;
2668 CK_OBJECT_HANDLE own_prekey;
2669 CK_OBJECT_HANDLE initiator_identity;
2670 CK_OBJECT_HANDLE own_public_identity;
2671 CK_BBOOL bEncryptedHeader;
2672 CK ULONG eCurve;
2673 CK_MECHANISM_TYPE aeadMechanism;
2674 CK_X2RATCHET_KDF_TYPE kdfMechanism;
2675 } CK_X2RATCHET_RESPOND_PARAMS;
```

2676 The fields of the structure have the following meanings:

|      |                            |                                                                          |
|------|----------------------------|--------------------------------------------------------------------------|
| 2678 | <i>sk</i>                  | <i>shared secret with the Initiator</i>                                  |
| 2679 | <i>own_prekey</i>          | <i>Own Prekey pair that the Initiator used</i>                           |
| 2680 | <i>initiator_identity</i>  | <i>Initiators public identity key used</i>                               |
| 2681 | <i>own_public_identity</i> | <i>as used in the prekey bundle by the initiator in the X3DH</i>         |
| 2682 | <i>bEncryptedHeader</i>    | <i>whether the headers are encrypted</i>                                 |
| 2683 | <i>eCurve</i>              | <i>255 for curve 25519 or 448 for curve 448</i>                          |
| 2684 | <i>aeadMechanism</i>       | <i>a mechanism supporting AEAD encryption</i>                            |
| 2685 | <i>kdfMechanism</i>        | <i>a Key Derivation Mechanism, such as</i><br><i>CKD_BLAKE2B_512_KDF</i> |

## 2687 2.6.4 Double Ratchet Encryption mechanism

2688 The Double Ratchet encryption mechanism, denoted **CKM\_X2RATCHET\_ENCRYPT** and  
 2689 **CKM\_X2RATCHET\_DECRYPT**, are a mechanisms for single part encryption and decryption based on  
 2690 the Double Ratchet and its underlying AEAD cipher.

## 2691 2.6.5 Double Ratchet parameters

- 2692 • **CK\_X2RATCHET\_KDF\_TYPE, CK\_X2RATCHET\_KDF\_TYPE\_PTR**

2693 **CK\_X2RATCHET\_KDF\_TYPE** is used to indicate the Key Derivation Function (KDF) applied to derive  
 2694 keying data from a shared secret. The key derivation function will be used by the X key derivation  
 2695 scheme. It is defined as follows:

```
2696 typedef CK ULONG CK_X2RATCHET_KDF_TYPE;
```

2698 The following table lists the defined functions.

2699 *Table 66, X2RATCHET: Key Derivation Functions*

| Source Identifier   |
|---------------------|
| CKD_NULL            |
| CKD_BLAKE2B_256_KDF |
| CKD_BLAKE2B_512_KDF |
| CKD_SHA3_256_KDF    |
| CKD_SHA256_KDF      |
| CKD_SHA3_512_KDF    |
| CKD_SHA512_KDF      |

2700

## 2701 **2.7 Wrapping/unwrapping private keys**

2702 Cryptoki Versions 2.01 and up allow the use of secret keys for wrapping and unwrapping RSA private  
2703 keys, Diffie-Hellman private keys, X9.42 Diffie-Hellman private keys, EC (also related to ECDSA) private  
2704 keys and DSA private keys. For wrapping, a private key is BER-encoded according to PKCS #8's  
2705 PrivateKeyInfo ASN.1 type. PKCS #8 requires an algorithm identifier for the type of the private key. The  
2706 object identifiers for the required algorithm identifiers are as follows:

```
2707 rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }
2708
2709 dhKeyAgreement OBJECT IDENTIFIER ::= { pkcs-3 1 }
2710
2711 dhpublicnumber OBJECT IDENTIFIER ::= { iso(1) member-body(2)
2712 us(840) ansi-x942(10046) number-type(2) 1 }
2713
2714 id-ecPublicKey OBJECT IDENTIFIER ::= { iso(1) member-body(2)
2715 us(840) ansi-x9-62(10045) publicKeyType(2) 1 }
2716
2717 id-dsa OBJECT IDENTIFIER ::= {
2718 iso(1) member-body(2) us(840) x9-57(10040) x9cm(4) 1 }
2719
2720 where
2721 pkcs-1 OBJECT IDENTIFIER ::= {
2722 iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) 1 }
2723
2724 pkcs-3 OBJECT IDENTIFIER ::= {
2725 iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) 3 }
2726
2727 These parameters for the algorithm identifiers have the
2728 following types, respectively:
2729 NULL
2730
2731 DHParameter ::= SEQUENCE {
2732 prime INTEGER, -- p
2733 base INTEGER, -- g
2734 privateValueLength INTEGER OPTIONAL
```

```

2735 }
2736
2737 DomainParameters ::= SEQUENCE {
2738 prime INTEGER, -- p
2739 base INTEGER, -- g
2740 subprime INTEGER, -- q
2741 cofactor INTEGER OPTIONAL, -- j
2742 validationParms ValidationParms OPTIONAL
2743 }
2744
2745 ValidationParms ::= SEQUENCE {
2746 Seed BIT STRING, -- seed
2747 PGenCounter INTEGER -- parameter verification
2748 }
2749
2750 Parameters ::= CHOICE {
2751 ecParameters ECParameters,
2752 namedCurve CURVES.&id({CurveNames}),
2753 implicitlyCA NULL
2754 }
2755
2756 Dss_Parms ::= SEQUENCE {
2757 p INTEGER,
2758 q INTEGER,
2759 g INTEGER
2760 }

```

2761 For the X9.42 Diffie-Hellman domain parameters, the **cofactor** and the **validationParms** optional fields  
 2762 should not be used when wrapping or unwrapping X9.42 Diffie-Hellman private keys since their values  
 2763 are not stored within the token.

2764 For the EC domain parameters, the use of **namedCurve** is recommended over the choice  
 2765 **ecParameters**. The choice **implicitlyCA** must not be used in Cryptoki.

2766 Within the PrivateKeyInfo type:

- 2767 • RSA private keys are BER-encoded according to PKCS #1's RSAPrivateKey ASN.1 type. This type  
 2768 requires values to be present for *all* the attributes specific to Cryptoki's RSA private key objects. In  
 2769 other words, if a Cryptoki library does not have values for an RSA private key's **CKA\_MODULUS**,  
 2770 **CKA\_PUBLIC\_EXPONENT**, **CKA\_PRIVATE\_EXPONENT**, **CKA\_PRIME\_1**, **CKA\_PRIME\_2**,  
 2771 **CKA\_EXPONENT\_1**, **CKA\_EXPONENT\_2**, and **CKA\_COEFFICIENT** values, it must not create an  
 2772 RSAPrivateKey BER-encoding of the key, and so it must not prepare it for wrapping.
- 2773 • Diffie-Hellman private keys are represented as BER-encoded ASN.1 type INTEGER.
- 2774 • X9.42 Diffie-Hellman private keys are represented as BER-encoded ASN.1 type INTEGER.
- 2775 • EC (also related with ECDSA) private keys are BER-encoded according to SECG SEC 1  
 2776 **ECPrivateKey** ASN.1 type:

```

2777 ECPrivateKey ::= SEQUENCE {
2778 Version INTEGER { ecPrivkeyVer1(1) }
2779 (ecPrivkeyVer1),
2780 privateKey OCTET STRING,
2781 parameters [0] Parameters OPTIONAL,

```

```
2783 publicKey [1] BIT STRING OPTIONAL
2784 }
2785
```

2786 Since the EC domain parameters are placed in the PKCS #8's privateKeyAlgorithm field, the optional  
2787 **parameters** field in an ECPrivateKey must be omitted. A Cryptoki application must be able to  
2788 unwrap an ECPrivateKey that contains the optional **publicKey** field; however, what is done with this  
2789 **publicKey** field is outside the scope of Cryptoki.

- DSA private keys are represented as BER-encoded ASN.1 type INTEGER.

2791 Once a private key has been BER-encoded as a PrivateKeyInfo type, the resulting string of bytes is  
2792 encrypted with the secret key. This encryption must be done in CBC mode with PKCS padding.

2793 Unwrapping a wrapped private key undoes the above procedure. The CBC-encrypted ciphertext is  
2794 decrypted, and the PKCS padding is removed. The data thereby obtained are parsed as a  
2795 PrivateKeyInfo type, and the wrapped key is produced. An error will result if the original wrapped key  
2796 does not decrypt properly, or if the decrypted unpadded data does not parse properly, or its type does not  
2797 match the key type specified in the template for the new key. The unwrapping mechanism contributes  
2798 only those attributes specified in the PrivateKeyInfo type to the newly-unwrapped key; other attributes  
2799 must be specified in the template, or will take their default values.

2800 Earlier drafts of PKCS #11 Version 2.0 and Version 2.01 used the object identifier

```
2801 DSA OBJECT IDENTIFIER ::= { algorithm 12 }
2802 algorithm OBJECT IDENTIFIER ::= {
2803 iso(1) identifier-organization(3) oiw(14) secsig(3)
2804 algorithm(2) }
```

2805

2806 with associated parameters

```
2807 DSAParameters ::= SEQUENCE {
2808 prime1 INTEGER, -- modulus p
2809 prime2 INTEGER, -- modulus q
2810 base INTEGER -- base g
2811 }
```

2812

2813 for wrapping DSA private keys. Note that although the two structures for holding DSA domain  
2814 parameters appear identical when instances of them are encoded, the two corresponding object  
2815 identifiers are different.

## 2816 2.8 Generic secret key

2817 *Table 67, Generic Secret Key Mechanisms vs. Functions*

| Mechanism                  | Functions         |               |                      |        |                    |               |        |
|----------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                            | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_GENERIC_SECRET_KEY_GEN |                   |               |                      |        | ✓                  |               |        |

### 2818 2.8.1 Definitions

2819 This section defines the key type "CKK\_GENERIC\_SECRET" for type CK\_KEY\_TYPE as used in the  
2820 CKA\_KEY\_TYPE attribute of key objects.

2821 Mechanisms:  
 2822       CKM\_GENERIC\_SECRET\_KEY\_GEN

2823 **2.8.2 Generic secret key objects**

2824 Generic secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_GENERIC\_SECRET**) hold  
 2825 generic secret keys. These keys do not support encryption or decryption; however, other keys can be  
 2826 derived from them and they can be used in HMAC operations. The following table defines the generic  
 2827 secret key object attributes, in addition to the common attributes defined for this object class:  
 2828 These key types are used in several of the mechanisms described in this section.

2829 *Table 68, Generic Secret Key Object Attributes*

| Attribute                    | Data type  | Meaning                      |
|------------------------------|------------|------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup> | Byte array | Key value (arbitrary length) |
| CKA_VALUE_LEN <sup>2,3</sup> | CK ULONG   | Length in bytes of key value |

2830 - Refer to [PKCS11-Base] table 11 for footnotes

2831 The following is a sample template for creating a generic secret key object:

```
2832 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
2833 CK_KEY_TYPE keyType = CKK_GENERIC_SECRET;
2834 CK_UTF8CHAR label[] = "A generic secret key object";
2835 CK_BYTE value[] = { ... };
2836 CK_BBOOL true = CK_TRUE;
2837 CK_ATTRIBUTE template[] = {
2838 { CKA_CLASS, &class, sizeof(class) },
2839 { CKA_KEY_TYPE, &keyType, sizeof(keyType) },
2840 { CKA_TOKEN, &true, sizeof(true) },
2841 { CKA_LABEL, label, sizeof(label)-1 },
2842 { CKA_DERIVE, &true, sizeof(true) },
2843 { CKA_VALUE, value, sizeof(value) }
2844 };
```

2845

2846 CKA\_CHECK\_VALUE: The value of this attribute is derived from the key object by taking the first three  
 2847 bytes of the SHA-1 hash of the generic secret key object's CKA\_VALUE attribute.

### 2848 **2.8.3 Generic secret key generation**

2849 The generic secret key generation mechanism, denoted **CKM\_GENERIC\_SECRET\_KEY\_GEN**, is used  
 2850 to generate generic secret keys. The generated keys take on any attributes provided in the template  
 2851 passed to the **C\_GenerateKey** call, and the **CKA\_VALUE\_LEN** attribute specifies the length of the key  
 2852 to be generated.

2853 It does not have a parameter.

2854 The template supplied must specify a value for the **CKA\_VALUE\_LEN** attribute. If the template specifies  
 2855 an object type and a class, they must have the following values:

```
2856 CK_OBJECT_CLASS = CKO_SECRET_KEY;
2857 CK_KEY_TYPE = CKK_GENERIC_SECRET;
```

2858 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 2859 specify the supported range of key sizes, in bits.

## 2860 2.9 HMAC mechanisms

2861 Refer to **RFC2104** and **FIPS 198** for HMAC algorithm description. The HMAC secret key shall correspond  
2862 to the PKCS11 generic secret key type or the mechanism specific key types (see mechanism definition).  
2863 Such keys, for use with HMAC operations can be created using C\_CreateObject or C\_GenerateKey.

2864 The RFC also specifies test vectors for the various hash function based HMAC mechanisms described in  
2865 the respective hash mechanism descriptions. The RFC should be consulted to obtain these test vectors.

### 2866 2.9.1 General block cipher mechanism parameters

- 2867 • **CK\_MAC\_GENERAL\_PARAMS; CK\_MAC\_GENERAL\_PARAMS\_PTR**

2868 **CK\_MAC\_GENERAL\_PARAMS** provides the parameters to the general-length MACing mechanisms of  
2869 the DES, DES3 (triple-DES), AES, Camellia, SEED, and ARIA ciphers. It also provides the parameters to  
2870 the general-length HMACing mechanisms (i.e.,SHA-1, SHA-256, SHA-384, SHA-512, and SHA-512/T  
2871 family) and the two SSL 3.0 MACing mechanisms, (i.e., MD5 and SHA-1). It holds the length of the MAC  
2872 that these mechanisms produce. It is defined as follows:

```
2873 typedef CK ULONG CK_MAC_GENERAL_PARAMS;
```

2875 **CK\_MAC\_GENERAL\_PARAMS\_PTR** is a pointer to a **CK\_MAC\_GENERAL\_PARAMS**.

## 2876 2.10 AES

2877 For the Advanced Encryption Standard (AES) see [FIPS PUB 197].

2878 *Table 69, AES Mechanisms vs. Functions*

| Mechanism           | Functions         |               |                      |        |                    |               |        |
|---------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                     | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_AES_KEY_GEN     |                   |               |                      |        | ✓                  |               |        |
| CKM_AES_ECB         | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_AES_CBC         | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_AES_CBC_PAD     | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_AES_MAC_GENERAL |                   | ✓             |                      |        |                    |               |        |
| CKM_AES_MAC         |                   | ✓             |                      |        |                    |               |        |
| CKM_AES_OFB         | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_AES_CFB64       | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_AES_CFB8        | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_AES_CFB128      | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_AES_CFB1        | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_AES_XCBC_MAC    |                   | ✓             |                      |        |                    |               |        |
| CKM_AES_XCBC_MAC_96 |                   | ✓             |                      |        |                    |               |        |

### 2879 2.10.1 Definitions

2880 This section defines the key type "CKK\_AES" for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE  
2881 attribute of key objects.

2882 Mechanisms:

2883 CKM\_AES\_KEY\_GEN

2884 CKM\_AES\_ECB

2885 CKM\_AES\_CBC

2886 CKM\_AES\_MAC

2887 CKM\_AES\_MAC\_GENERAL

2888 CKM\_AES\_CBC\_PAD

2889 CKM\_AES\_OFB

2890 CKM\_AES\_CFB64

2891 CKM\_AES\_CFB8

2892 CKM\_AES\_CFB128

2893 CKM\_AES\_CFB1

2894 CKM\_AES\_XCBC\_MAC

2895 CKM\_AES\_XCBC\_MAC\_96

## 2896 2.10.2 AES secret key objects

2897 AES secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_AES**) hold AES keys. The  
 2898 following table defines the AES secret key object attributes, in addition to the common attributes defined  
 2899 for this object class:

2900 *Table 70, AES Secret Key Object Attributes*

| Attribute                      | Data type  | Meaning                         |
|--------------------------------|------------|---------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup>   | Byte array | Key value (16, 24, or 32 bytes) |
| CKA_VALUE_LEN <sup>2,3,6</sup> | CK ULONG   | Length in bytes of key value    |

2901 - Refer to [PKCS11-Base] table 11 for footnotes

2902 The following is a sample template for creating an AES secret key object:

```
2903 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
2904 CK_KEY_TYPE keyType = CKK_AES;
2905 CK_UTF8CHAR label[] = "An AES secret key object";
2906 CK_BYTE value[] = { ... };
2907 CK_BBOOL true = CK_TRUE;
2908 CK_ATTRIBUTE template[] = {
2909 {CKA_CLASS, &class, sizeof(class)},
2910 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
2911 {CKA_TOKEN, &true, sizeof(true)},
2912 {CKA_LABEL, label, sizeof(label)-1},
2913 {CKA_ENCRYPT, &true, sizeof(true)},
2914 {CKA_VALUE, value, sizeof(value)}}
2915 };
```

2916

2917 CKA\_CHECK\_VALUE: The value of this attribute is derived from the key object by taking the first three  
 2918 bytes of the ECB encryption of a single block of null (0x00) bytes, using the default cipher associated with  
 2919 the key type of the secret key object.

2920 **2.10.3 AES key generation**

2921 The AES key generation mechanism, denoted **CKM\_AES\_KEY\_GEN**, is a key generation mechanism for  
2922 NIST's Advanced Encryption Standard.

2923 It does not have a parameter.

2924 The mechanism generates AES keys with a particular length in bytes, as specified in the  
2925 **CKA\_VALUE\_LEN** attribute of the template for the key.

2926 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
2927 key. Other attributes supported by the AES key type (specifically, the flags indicating which functions the  
2928 key supports) may be specified in the template for the key, or else are assigned default initial values.

2929 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2930 specify the supported range of AES key sizes, in bytes.

2931 **2.10.4 AES-ECB**

2932 AES-ECB, denoted **CKM\_AES\_ECB**, is a mechanism for single- and multiple-part encryption and  
2933 decryption; key wrapping; and key unwrapping, based on NIST Advanced Encryption Standard and  
2934 electronic codebook mode.

2935 It does not have a parameter.

2936 This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to  
2937 wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the  
2938 **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to block size minus  
2939 one null bytes so that the resulting length is a multiple of the block size. The output data is the same  
2940 length as the padded input data. It does not wrap the key type, key length, or any other information about  
2941 the key; the application must convey these separately.

2942 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
2943 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the  
2944 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
2945 attribute of the new key; other attributes required by the key type must be specified in the template.

2946 Constraints on key types and the length of data are summarized in the following table:

2947 *Table 71, AES-ECB: Key And Data Length*

| Function    | Key type | Input length           | Output length                                                     | Comments      |
|-------------|----------|------------------------|-------------------------------------------------------------------|---------------|
| C_Encrypt   | AES      | multiple of block size | same as input length                                              | no final part |
| C_Decrypt   | AES      | multiple of block size | same as input length                                              | no final part |
| C_WrapKey   | AES      | any                    | input length rounded up to multiple of block size                 |               |
| C_UnwrapKey | AES      | multiple of block size | determined by type of key being unwrapped or <b>CKA_VALUE_LEN</b> |               |

2948 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2949 specify the supported range of AES key sizes, in bytes.

2950 **2.10.5 AES-CBC**

2951 AES-CBC, denoted **CKM\_AES\_CBC**, is a mechanism for single- and multiple-part encryption and  
2952 decryption; key wrapping; and key unwrapping, based on NIST's Advanced Encryption Standard and  
2953 cipher-block chaining mode.

2954 It has a parameter, a 16-byte initialization vector.

2955 This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to  
2956 wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the  
2957 **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to block size minus  
2958 one null bytes so that the resulting length is a multiple of the block size. The output data is the same  
2959 length as the padded input data. It does not wrap the key type, key length, or any other information about  
2960 the key; the application must convey these separately.

2961 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
2962 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the  
2963 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
2964 attribute of the new key; other attributes required by the key type must be specified in the template.

2965 Constraints on key types and the length of data are summarized in the following table:

2966 *Table 72, AES-CBC: Key And Data Length*

| Function    | Key type | Input length           | Output length                                                     | Comments      |
|-------------|----------|------------------------|-------------------------------------------------------------------|---------------|
| C_Encrypt   | AES      | multiple of block size | same as input length                                              | no final part |
| C_Decrypt   | AES      | multiple of block size | same as input length                                              | no final part |
| C_WrapKey   | AES      | any                    | input length rounded up to multiple of the block size             |               |
| C_UnwrapKey | AES      | multiple of block size | determined by type of key being unwrapped or <b>CKA_VALUE_LEN</b> |               |

2967 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2968 specify the supported range of AES key sizes, in bytes.

## 2969 **2.10.6 AES-CBC with PKCS padding**

2970 AES-CBC with PKCS padding, denoted **CKM\_AES\_CBC\_PAD**, is a mechanism for single- and multiple-  
2971 part encryption and decryption; key wrapping; and key unwrapping, based on NIST's Advanced  
2972 Encryption Standard; cipher-block chaining mode; and the block cipher padding method detailed in PKCS  
2973 #7.

2974 It has a parameter, a 16-byte initialization vector.

2975 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the  
2976 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified  
2977 for the **CKA\_VALUE\_LEN** attribute.

2978 In addition to being able to wrap and unwrap secret keys, this mechanism can wrap and unwrap RSA,  
2979 Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see Section 2.7  
2980 for details). The entries in the table below for data length constraints when wrapping and unwrapping  
2981 keys do not apply to wrapping and unwrapping private keys.

2982 Constraints on key types and the length of data are summarized in the following table:

2983 Table 73, AES-CBC with PKCS Padding: Key And Data Length

| Function    | Key type | Input length           | Output length                                              |
|-------------|----------|------------------------|------------------------------------------------------------|
| C_Encrypt   | AES      | any                    | input length rounded up to multiple of the block size      |
| C_Decrypt   | AES      | multiple of block size | between 1 and block size bytes shorter than input length   |
| C_WrapKey   | AES      | any                    | input length rounded up to multiple of the block size      |
| C_UnwrapKey | AES      | multiple of block size | between 1 and block length bytes shorter than input length |

2984 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
2985 specify the supported range of AES key sizes, in bytes.

## 2986 2.10.7 AES-OFB

2987 AES-OFB, denoted CKM\_AES\_OFB. It is a mechanism for single and multiple-part encryption and  
2988 decryption with AES. AES-OFB mode is described in [NIST sp800-38a].

2989 It has a parameter, an initialization vector for this mode. The initialization vector has the same length as  
2990 the block size.

2991 Constraints on key types and the length of data are summarized in the following table:  
2992

2994 Table 74, AES-OFB: Key And Data Length

| Function  | Key type | Input length | Output length        | Comments      |
|-----------|----------|--------------|----------------------|---------------|
| C_Encrypt | AES      | any          | same as input length | no final part |
| C_Decrypt | AES      | any          | same as input length | no final part |

2995 For this mechanism the CK\_MECHANISM\_INFO structure is as specified for CBC mode.

## 2996 2.10.8 AES-CFB

2997 Cipher AES has a cipher feedback mode, AES-CFB, denoted CKM\_AES\_CFB8, CKM\_AES\_CFB64, and  
2998 CKM\_AES\_CFB128. It is a mechanism for single and multiple-part encryption and decryption with AES.  
2999 AES-OFB mode is described [NIST sp800-38a].

3000 It has a parameter, an initialization vector for this mode. The initialization vector has the same length as  
3001 the block size.

3002 Constraints on key types and the length of data are summarized in the following table:  
3003

3005 Table 75, AES-CFB: Key And Data Length

| Function  | Key type | Input length | Output length        | Comments      |
|-----------|----------|--------------|----------------------|---------------|
| C_Encrypt | AES      | any          | same as input length | no final part |
| C_Decrypt | AES      | any          | same as input length | no final part |

3006 For this mechanism the CK\_MECHANISM\_INFO structure is as specified for CBC mode.

3007 **2.10.9 General-length AES-MAC**

3008 General-length AES-MAC, denoted **CKM\_AES\_MAC\_GENERAL**, is a mechanism for single- and  
3009 multiple-part signatures and verification, based on NIST Advanced Encryption Standard as defined in  
3010 FIPS PUB 197 and data authentication as defined in FIPS PUB 113.

3011 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS** structure, which specifies the output length  
3012 desired from the mechanism.

3013 The output bytes from this mechanism are taken from the start of the final AES cipher block produced in  
3014 the MACing process.

3015 Constraints on key types and the length of data are summarized in the following table:

3016 *Table 76, General-length AES-MAC: Key And Data Length*

| Function | Key type | Data length | Signature length                         |
|----------|----------|-------------|------------------------------------------|
| C_Sign   | AES      | any         | 1-block size, as specified in parameters |
| C_Verify | AES      | any         | 1-block size, as specified in parameters |

3017 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
3018 specify the supported range of AES key sizes, in bytes.

3019 **2.10.10 AES-MAC**

3020 AES-MAC, denoted by **CKM\_AES\_MAC**, is a special case of the general-length AES-MAC mechanism.  
3021 AES-MAC always produces and verifies MACs that are half the block size in length.

3022 It does not have a parameter.

3023 Constraints on key types and the length of data are summarized in the following table:

3024 *Table 77, AES-MAC: Key And Data Length*

| Function | Key type | Data length | Signature length       |
|----------|----------|-------------|------------------------|
| C_Sign   | AES      | Any         | ½ block size (8 bytes) |
| C_Verify | AES      | Any         | ½ block size (8 bytes) |

3025 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
3026 specify the supported range of AES key sizes, in bytes.

3027 **2.10.11 AES-XCBC-MAC**

3028 AES-XCBC-MAC, denoted **CKM\_AES\_XCBC\_MAC**, is a mechanism for single and multiple part  
3029 signatures and verification; based on NIST's Advanced Encryption Standard and [RFC 3566].

3030 It does not have a parameter.

3031 Constraints on key types and the length of data are summarized in the following table:

3032 *Table 78, AES-XCBC-MAC: Key And Data Length*

| Function | Key type | Data length | Signature length |
|----------|----------|-------------|------------------|
| C_Sign   | AES      | Any         | 16 bytes         |
| C_Verify | AES      | Any         | 16 bytes         |

3033 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
3034 specify the supported range of AES key sizes, in bytes.

3035 **2.10.12 AES-XCBC-MAC-96**

3036 AES-XCBC-MAC-96, denoted **CKM\_AES\_XCBC\_MAC\_96**, is a mechanism for single and multiple part  
3037 signatures and verification; based on NIST's Advanced Encryption Standard and [RFC 3566].

3038 It does not have a parameter.

3039 Constraints on key types and the length of data are summarized in the following table:

3040 *Table 79, AES-XCBC-MAC: Key And Data Length*

| Function | Key type | Data length | Signature length |
|----------|----------|-------------|------------------|
| C_Sign   | AES      | Any         | 12 bytes         |
| C_Verify | AES      | Any         | 12 bytes         |

3041 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
3042 specify the supported range of AES key sizes, in bytes.

## 3043 **2.11 AES with Counter**

3044 *Table 80, AES with Counter Mechanisms vs. Functions*

| Mechanism   | Functions         |               |                      |        |                    |               |        |
|-------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|             | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_AES_CTR | ✓                 |               |                      |        |                    | ✓             |        |

### 3045 **2.11.1 Definitions**

3046 Mechanisms:

3047 CKM\_AES\_CTR

### 3048 **2.11.2 AES with Counter mechanism parameters**

#### 3049 ♦ **CK\_AES\_CTR\_PARAMS; CK\_AES\_CTR\_PARAMS\_PTR**

3050 **CK\_AES\_CTR\_PARAMS** is a structure that provides the parameters to the **CKM\_AES\_CTR** mechanism.  
3051 It is defined as follows:

```
3052 typedef struct CK_AES_CTR_PARAMS {
3053 CK ULONG ulCounterBits;
3054 CK BYTE cb[16];
3055 } CK_AES_CTR_PARAMS;
```

3057 ulCounterBits specifies the number of bits in the counter block (cb) that shall be incremented. This  
3058 number shall be such that 0 < ulCounterBits <= 128. For any values outside this range the mechanism  
3059 shall return **CKR\_MECHANISM\_PARAM\_INVALID**.

3060 It's up to the caller to initialize all of the bits in the counter block including the counter bits. The counter  
3061 bits are the least significant bits of the counter block (cb). They are a big-endian value usually starting  
3062 with 1. The rest of 'cb' is for the nonce, and maybe an optional IV.

3063 E.g. as defined in [RFC 3686]:

```
3064 0 1 2 3
3065 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
3066 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3067 | Nonce |
3068 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3069 | Initialization Vector (IV) |
3070 |
3071 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

3072 | Block Counter |  
3073 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

3074

3075 This construction permits each packet to consist of up to  $2^{32}-1$  blocks = 4,294,967,295 blocks =  
3076 68,719,476,720 octets.

3077 CK\_AES\_CTR\_PARAMS\_PTR is a pointer to a CK\_AES\_CTR\_PARAMS.

### 3078 2.11.3 AES with Counter Encryption / Decryption

3079 Generic AES counter mode is described in NIST Special Publication 800-38A and in RFC 3686. These  
3080 describe encryption using a counter block which may include a nonce to guarantee uniqueness of the  
3081 counter block. Since the nonce is not incremented, the mechanism parameter must specify the number of  
3082 counter bits in the counter block.

3083 The block counter is incremented by 1 after each block of plaintext is processed. There is no support for  
3084 any other increment functions in this mechanism.

3085 If an attempt to encrypt/decrypt is made which will cause an overflow of the counter block's counter bits,  
3086 then the mechanism shall return CKR\_DATA\_LEN\_RANGE. Note that the mechanism should allow the  
3087 final post increment of the counter to overflow (if it implements it this way) but not allow any further  
3088 processing after this point. E.g. if ulCounterBits = 2 and the counter bits start as 1 then only 3 blocks of  
3089 data can be processed.

3090

## 3091 2.12 AES CBC with Cipher Text Stealing CTS

3092 Ref [NIST AES CTS]

3093 This mode allows unpadded data that has length that is not a multiple of the block size to be encrypted to  
3094 the same length of cipher text.

3095 *Table 81, AES CBC with Cipher Text Stealing CTS Mechanisms vs. Functions*

| Mechanism   | Functions         |               |                      |        |                    |               |        |
|-------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|             | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_AES_CTS | ✓                 |               |                      |        |                    | ✓             |        |

### 3096 2.12.1 Definitions

3097 Mechanisms:

3098 CKM\_AES\_CTS

### 3099 2.12.2 AES CTS mechanism parameters

3100 It has a parameter, a 16-byte initialization vector.

3101 Table 82, AES-CTS: Key And Data Length

| Function  | Key type | Input length                      | Output length        |  |  | Comments      |
|-----------|----------|-----------------------------------|----------------------|--|--|---------------|
| C_Encrypt | AES      | Any, $\geq$ block size (16 bytes) | same as input length |  |  | no final part |
| C_Decrypt | AES      | any, $\geq$ block size (16 bytes) | same as input length |  |  | no final part |

3102

## 2.13 Additional AES Mechanisms

3104 Table 83, Additional AES Mechanisms vs. Functions

| Mechanism    | Functions         |               |                      |        |                   |               |        |
|--------------|-------------------|---------------|----------------------|--------|-------------------|---------------|--------|
|              | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/Key Pair | Wrap & Unwrap | Derive |
| CKM_AES_GCM  | ✓                 |               |                      |        |                   | ✓             |        |
| CKM_AES_CCM  | ✓                 |               |                      |        |                   | ✓             |        |
| CKM_AES_GMAC |                   | ✓             |                      |        |                   |               |        |

3105

### 2.13.1 Definitions

3107 Mechanisms:

3108 CKM\_AES\_GCM

3109 CKM\_AES\_CCM

3110 CKM\_AES\_GMAC

3111 Generator Functions:

3112 CKG\_NO\_GENERATE

3113 CKG\_GENERATE

3114 CKG\_GENERATE\_COUNTER

3115 CKG\_GENERATE\_RANDOM

### 2.13.2 AES-GCM Authenticated Encryption / Decryption

3117 Generic GCM mode is described in [GCM]. To set up for AES-GCM use the following process, where  $K$  (key) and  $AAD$  (additional authenticated data) are as described in [GCM]. AES-GCM uses CK\_GCM\_PARAMS for Encrypt, Decrypt and CK\_GCM\_MESSAGE\_PARAMS for MessageEncrypt and MessageDecrypt.

3121 Encrypt:

- 3122 • Set the IV length  $ulvLen$  in the parameter block.
- 3123 • Set the IV data  $pIV$  in the parameter block.
- 3124 • Set the AAD data  $pAAD$  and size  $ulAADLen$  in the parameter block.  $pAAD$  may be NULL if  $ulAADLen$  is 0.
- 3126 • Set the tag length  $ulTagBits$  in the parameter block.
- 3127 • Call C\_EncryptInit() for **CKM\_AES\_GCM** mechanism with parameters and key  $K$ .

- 3128     • Call C\_Encrypt(), or C\_EncryptUpdate()<sup>\*4</sup> C\_EncryptFinal(), for the plaintext obtaining ciphertext  
 3129       and authentication tag output.
- 3130   Decrypt:
- 3131     • Set the IV length *ullvLen* in the parameter block.
- 3132     • Set the IV data *pIV* in the parameter block.
- 3133     • Set the AAD data *pAAD* and size *ulAADLen* in the parameter block. *pAAD* may be NULL if  
 3134       *ulAADLen* is 0.
- 3135     • Set the tag length *ulTagBits* in the parameter block.
- 3136     • Call C\_DecryptInit() for **CKM\_AES\_GCM** mechanism with parameters and key *K*.
- 3137     • Call C\_Decrypt(), or C\_DecryptUpdate()<sup>\*1</sup> C\_DecryptFinal(), for the ciphertext, including the  
 3138       appended tag, obtaining plaintext output. Note: since **CKM\_AES\_GCM** is an AEAD cipher, no  
 3139       data should be returned until C\_Decrypt() or C\_DecryptFinal().
- 3140   MessageEncrypt:
- 3141     • Set the IV length *ullvLen* in the parameter block.
- 3142     • Set *pIV* to hold the IV data returned from C\_EncryptMessage() and C\_EncryptMessageBegin(). If  
 3143       *ullvFixedBits* is not zero, then the most significant bits of *pIV* contain the fixed IV. If *ivGenerator* is  
 3144       set to CKG\_NO\_GENERATE, *pIV* is an input parameter with the full IV.
- 3145     • Set the *ullvFixedBits* and *ivGenerator* fields in the parameter block.
- 3146     • Set the tag length *ulTagBits* in the parameter block.
- 3147     • Set *pTag* to hold the tag data returned from C\_EncryptMessage() or the final  
 3148       C\_EncryptMessageNext().
- 3149     • Call C\_MessageEncryptInit() for **CKM\_AES\_GCM** mechanism key *K*.
- 3150     • Call C\_EncryptMessage(), or C\_EncryptMessageBegin() followed by C\_EncryptMessageNext()<sup>\*5</sup>.  
 3151       The mechanism parameter is passed to all three of these functions.
- 3152     • Call C\_MessageEncryptFinal() to close the message decryption.
- 3153   MessageDecrypt:
- 3154     • Set the IV length *ullvLen* in the parameter block.
- 3155     • Set the IV data *pIV* in the parameter block.
- 3156     • The *ullvFixedBits* and *ivGenerator* fields are ignored.
- 3157     • Set the tag length *ulTagBits* in the parameter block.
- 3158     • Set the tag data *pTag* in the parameter block before C\_DecryptMessage() or the final  
 3159       C\_DecryptMessageNext().
- 3160     • Call C\_MessageDecryptInit() for **CKM\_AES\_GCM** mechanism key *K*.
- 3161     • Call C\_DecryptMessage(), or C\_DecryptMessageBegin followed by C\_DecryptMessageNext()<sup>\*6</sup>.  
 3162       The mechanism parameter is passed to all three of these functions.
- 3163     • Call C\_MessageDecryptFinal() to close the message decryption.

<sup>4</sup> <sup>\*\*</sup> indicates 0 or more calls may be made as required

<sup>5</sup> <sup>\*\*</sup> indicates 0 or more calls may be made as required

<sup>6</sup> <sup>\*\*</sup> indicates 0 or more calls may be made as required

3164 In *pIV* the least significant bit of the initialization vector is the rightmost bit. *ullvLen* is the length of the  
3165 initialization vector in bytes.

3166 On MessageEncrypt, the meaning of *ivGenerator* is as follows: CKG\_NO\_GENERATE means the IV is  
3167 passed in on MessageEncrypt and no internal IV generation is done. CKG\_GENERATE means that the  
3168 non-fixed portion of the IV is generated by the module internally. The generation method is not defined.  
3169 CKG\_GENERATE\_COUNTER means that the non-fixed portion of the IV is generated by the module  
3170 internally by use of an incrementing counter. CKG\_GENERATE\_RANDOM means that the non-fixed  
3171 portion of the IV is generated by the module internally using a PRNG. In any case the entire IV, including  
3172 the fixed portion, is returned in *pIV*.

3173 Modules must implement CKG\_GENERATE. Modules may also reject *ullvFixedBits* values which are too  
3174 large. Zero is always an acceptable value for *ullvFixedBits*.

3175 In Encrypt and Decrypt the tag is appended to the cipher text and the least significant bit of the tag is the  
3176 rightmost bit and the tag bits are the rightmost *ulTagBits* bits. In MessageEncrypt the tag is returned in  
3177 the *pTag* field of CK\_GCM\_MESSAGE\_PARAMS. In MessageDecrypt the tag is provided by the *pTag*  
3178 field of CK\_GCM\_MESSAGE\_PARAMS.

3179 The key type for *K* must be compatible with **CKM\_AES\_ECB** and the  
3180 C\_EncryptInit()/C\_DecryptInit()/C\_MessageEncryptInit()/C\_MessageDecryptInit() calls shall behave, with  
3181 respect to *K*, as if they were called directly with **CKM\_AES\_ECB**, *K* and NULL parameters.

### 3182 2.13.3 AES-CCM authenticated Encryption / Decryption

3183 For IPsec (RFC 4309) and also for use in ZFS encryption. Generic CCM mode is described in [RFC  
3184 3610].

3185 To set up for AES-CCM use the following process, where *K* (key), nonce and additional authenticated  
3186 data are as described in [RFC 3610]. AES-CCM uses CK\_CCM\_PARAMS for Encrypt and Decrypt, and  
3187 CK\_CCM\_MESSAGE\_PARAMS for MessageEncrypt and MessageDecrypt.

3188 Encrypt:

- 3189 • Set the message/data length *ulDataLen* in the parameter block.
- 3190 • Set the nonce length *ulNonceLen* and the nonce data *pNonce* in the parameter block.
- 3191 • Set the AAD data *pAAD* and size *ulAADLen* in the parameter block. *pAAD* may be NULL if  
3192 *ulAADLen* is 0.
- 3193 • Set the MAC length *ulMACLen* in the parameter block.
- 3194 • Call C\_EncryptInit() for **CKM\_AES\_CCM** mechanism with parameters and key *K*.
- 3195 • Call C\_Encrypt(), C\_EncryptUpdate(), or C\_EncryptFinal(), for the plaintext obtaining the final  
3196 ciphertext output and the MAC. The total length of data processed must be *ulDataLen*. The output  
3197 length will be *ulDataLen* + *ulMACLen*.

3198 Decrypt:

- 3199 • Set the message/data length *ulDataLen* in the parameter block. This length must not include the  
3200 length of the MAC that is appended to the cipher text.
- 3201 • Set the nonce length *ulNonceLen* and the nonce data *pNonce* in the parameter block.
- 3202 • Set the AAD data *pAAD* and size *ulAADLen* in the parameter block. *pAAD* may be NULL if  
3203 *ulAADLen* is 0.
- 3204 • Set the MAC length *ulMACLen* in the parameter block.
- 3205 • Call C\_DecryptInit() for **CKM\_AES\_CCM** mechanism with parameters and key *K*.
- 3206 • Call C\_Decrypt(), C\_DecryptUpdate(), or C\_DecryptFinal(), for the ciphertext, including the  
3207 appended MAC, obtaining plaintext output. The total length of data processed must be *ulDataLen*  
3208 + *ulMACLen*. Note: since **CKM\_AES\_CCM** is an AEAD cipher, no data should be returned until  
3209 C\_Decrypt() or C\_DecryptFinal().

3210     MessageEncrypt:

- 3211       • Set the message/data length *ulDataLen* in the parameter block.
- 3212       • Set the nonce length *ulNonceLen*.
- 3213       • Set *pNonce* to hold the nonce data returned from *C\_EncryptMessage()* and *C\_EncryptMessageBegin()*. If *ulNonceFixedBits* is not zero, then the most significant bits of *pNonce* contain the fixed nonce. If *nonceGenerator* is set to *CKG\_NO\_GENERATE*, *pNonce* is an input parameter with the full nonce.
- 3217       • Set the *ulNonceFixedBits* and *nonceGenerator* fields in the parameter block.
- 3218       • Set the MAC length *ulMACLen* in the parameter block.
- 3219       • Set *pMAC* to hold the MAC data returned from *C\_EncryptMessage()* or the final *C\_EncryptMessageNext()*.
- 3221       • Call *C\_MessageEncryptInit()* for **CKM\_AES\_CCM** mechanism key *K*.
- 3222       • Call *C\_EncryptMessage()*, or *C\_EncryptMessageBegin()* followed by *C\_EncryptMessageNext()*<sup>7</sup>. The mechanism parameter is passed to all three functions.
- 3224       • Call *C\_MessageEncryptFinal()* to close the message encryption.
- 3225       • The MAC is returned in *pMac* of the *CK\_CCM\_MESSAGE\_PARAMS* structure.

3226     MessageDecrypt:

- 3227       • Set the message/data length *ulDataLen* in the parameter block.
- 3228       • Set the nonce length *ulNonceLen* and the nonce data *pNonce* in the parameter block
- 3229       • The *ulNonceFixedBits* and *nonceGenerator* fields in the parameter block are ignored.
- 3230       • Set the MAC length *ulMACLen* in the parameter block.
- 3231       • Set the MAC data *pMAC* in the parameter block before *C\_DecryptMessage()* or the final *C\_DecryptMessageNext()*.
- 3233       • Call *C\_MessageDecryptInit()* for **CKM\_AES\_CCM** mechanism key *K*.
- 3234       • Call *C\_DecryptMessage()*, or *C\_DecryptMessageBegin()* followed by *C\_DecryptMessageNext()*<sup>8</sup>. The mechanism parameter is passed to all three functions.
- 3236       • Call *C\_MessageDecryptFinal()* to close the message decryption.

3237     In *pNonce* the least significant bit of the nonce is the rightmost bit. *ulNonceLen* is the length of the nonce in bytes.

3239     On MessageEncrypt, the meaning of *nonceGenerator* is as follows: *CKG\_NO\_GENERATE* means the nonce is passed in on MessageEncrypt and no internal MAC generation is done. *CKG\_GENERATE* means that the non-fixed portion of the nonce is generated by the module internally. The generation method is not defined. *CKG\_GENERATE\_COUNTER* means that the non-fixed portion of the nonce is generated by the module internally by use of an incrementing counter. *CKG\_GENERATE\_RANDOM* means that the non-fixed portion of the nonce is generated by the module internally using a PRNG. In any case the entire nonce, including the fixed portion, is returned in *pNonce*.

3246     Modules must implement *CKG\_GENERATE*. Modules may also reject *ulNonceFixedBits* values which are too large. Zero is always an acceptable value for *ulNonceFixedBits*.

---

<sup>7</sup> <sup>\*\*</sup> indicates 0 or more calls may be made as required

<sup>8</sup> <sup>\*\*</sup> indicates 0 or more calls may be made as required

3248 In Encrypt and Decrypt the MAC is appended to the cipher text and the least significant byte of the MAC  
3249 is the rightmost byte and the MAC bytes are the rightmost *ulMACLen* bytes. In MessageEncrypt the MAC  
3250 is returned in the *pMAC* field of CK\_CCM\_MESSAGE\_PARAMS. In MessageDecrypt the MAC is  
3251 provided by the *pMAC* field of CK\_CCM\_MESSAGE\_PARAMS.

3252 The key type for K must be compatible with **CKM\_AES\_ECB** and the  
3253 C\_EncryptInit()/C\_DecryptInit()/C\_MessageEncryptInit()/C\_MessageDecryptInit() calls shall behave, with  
3254 respect to K, as if they were called directly with **CKM\_AES\_ECB**, K and NULL parameters.

## 3255 2.13.4 AES-GMAC

3256 AES-GMAC, denoted **CKM\_AES\_GMAC**, is a mechanism for single and multiple-part signatures and  
3257 verification. It is described in NIST Special Publication 800-38D [GMAC]. GMAC is a special case of  
3258 GCM that authenticates only the Additional Authenticated Data (AAD) part of the GCM mechanism  
3259 parameters. When GMAC is used with C\_Sign or C\_Verify, pData points to the AAD. GMAC does not  
3260 use plaintext or ciphertext.

3261 The signature produced by GMAC, also referred to as a Tag, the tag's length is determined by the  
3262 CK\_GCM\_PARAMS field *ulTagBits*.

3263 The IV length is determined by the CK\_GCM\_PARAMS field *ullvLen*.

3264 Constraints on key types and the length of data are summarized in the following table:

3265 *Table 84, AES-GMAC: Key And Data Length*

| Function | Key type | Data length | Signature length             |
|----------|----------|-------------|------------------------------|
| C_Sign   | CKK_AES  | < 2^64      | Depends on param's ulTagBits |
| C_Verify | CKK_AES  | < 2^64      | Depends on param's ulTagBits |

3266 For this mechanism, the ulMinKeySize and ulMaxKeySize fields of the **CK\_MECHANISM\_INFO** structure  
3267 specify the supported range of AES key sizes, in bytes.

## 3268 2.13.5 AES GCM and CCM Mechanism parameters

### 3269 ◆ **CK\_GENERATOR\_FUNCTION**

3270 Functions to generate unique IVs and nonces.

```
3271 typedef CK ULONG CK_GENERATOR_FUNCTION;
```

### 3272 ◆ **CK\_GCM\_PARAMS; CK\_GCM\_PARAMS\_PTR**

3273 CK\_GCM\_PARAMS is a structure that provides the parameters to the CKM\_AES\_GCM mechanism  
3274 when used for Encrypt or Decrypt. It is defined as follows:

```
3275 typedef struct CK_GCM_PARAMS {
3276 CK_BYTE_PTR pIv;
3277 CK ULONG ulIvLen;
3278 CK ULONG ulIvBits;
3279 CK_BYTE_PTR pAAD;
3280 CK ULONG ulAADLen;
3281 CK ULONG ulTagBits;
3282 } CK_GCM_PARAMS;
```

3283

3284 The fields of the structure have the following meanings:

3285 *pIv* pointer to initialization vector

|      |                  |                                                                                                                                                                                                                                                                                  |
|------|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3286 | <i>ullvLen</i>   | <i>length of initialization vector in bytes. The length of the initialization vector can be any number between 1 and (2^32) - 1. 96-bit (12 byte) IV values can be processed more efficiently, so that length is recommended for situations in which efficiency is critical.</i> |
| 3287 |                  |                                                                                                                                                                                                                                                                                  |
| 3288 |                  |                                                                                                                                                                                                                                                                                  |
| 3289 |                  |                                                                                                                                                                                                                                                                                  |
| 3290 | <i>ullvBits</i>  | <i>length of initialization vector in bits. Do not use ullvBits to specify the length of the initialization vector, but ullvLen instead.</i>                                                                                                                                     |
| 3291 |                  |                                                                                                                                                                                                                                                                                  |
| 3292 | <i>pAAD</i>      | <i>pointer to additional authentication data. This data is authenticated but not encrypted.</i>                                                                                                                                                                                  |
| 3293 |                  |                                                                                                                                                                                                                                                                                  |
| 3294 | <i>ulAADLen</i>  | <i>length of pAAD in bytes. The length of the AAD can be any number between 0 and (2^32) - 1.</i>                                                                                                                                                                                |
| 3295 |                  |                                                                                                                                                                                                                                                                                  |
| 3296 | <i>ulTagBits</i> | <i>length of authentication tag (output following cipher text) in bits. Can be any value between 0 and 128.</i>                                                                                                                                                                  |
| 3297 |                  |                                                                                                                                                                                                                                                                                  |

3298 **CK\_GCM\_PARAMS\_PTR** is a pointer to a **CK\_GCM\_PARAMS**.

3299 ◆ **CK\_GCM\_MESSAGE\_PARAMS; CK\_GCM\_MESSAGE\_PARAMS\_PTR**

3300 CK\_GCM\_MESSAGE\_PARAMS is a structure that provides the parameters to the CKM\_AES\_GCM  
3301 mechanism when used for MessageEncrypt or MessageDecrypt. It is defined as follows:

```
3302 typedef struct CK_GCM_MESSAGE_PARAMS {
3303 CK_BYTE_PTR pIv;
3304 CK ULONG ulIvLen;
3305 CK ULONG ulIvFixedBits;
3306 CK_GENERATOR_FUNCTION ivGenerator;
3307 CK_BYTE_PTR pTag;
3308 CK ULONG ulTagBits;
3309 } CK_GCM_MESSAGE_PARAMS;
```

3310

3311 The fields of the structure have the following meanings:

|      |                      |                                                                                                                                                                                                                                                                                  |
|------|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3312 | <i>plv</i>           | <i>pointer to initialization vector</i>                                                                                                                                                                                                                                          |
| 3313 | <i>ullvLen</i>       | <i>length of initialization vector in bytes. The length of the initialization vector can be any number between 1 and (2^32) - 1. 96-bit (12 byte) IV values can be processed more efficiently, so that length is recommended for situations in which efficiency is critical.</i> |
| 3314 |                      |                                                                                                                                                                                                                                                                                  |
| 3315 |                      |                                                                                                                                                                                                                                                                                  |
| 3316 |                      |                                                                                                                                                                                                                                                                                  |
| 3317 | <i>ullvFixedBits</i> | <i>number of bits of the original IV to preserve when generating a new IV. These bits are counted from the Most significant bits (to the right).</i>                                                                                                                             |
| 3318 |                      |                                                                                                                                                                                                                                                                                  |
| 3319 |                      |                                                                                                                                                                                                                                                                                  |
| 3320 | <i>ivGenerator</i>   | <i>Function used to generate a new IV. Each IV must be unique for a given session.</i>                                                                                                                                                                                           |
| 3321 |                      |                                                                                                                                                                                                                                                                                  |
| 3322 | <i>pTag</i>          | <i>location of the authentication tag which is returned on MessageEncrypt, and provided on MessageDecrypt.</i>                                                                                                                                                                   |
| 3323 |                      |                                                                                                                                                                                                                                                                                  |
| 3324 | <i>ulTagBits</i>     | <i>length of authentication tag in bits. Can be any value between 0 and 128.</i>                                                                                                                                                                                                 |
| 3325 |                      |                                                                                                                                                                                                                                                                                  |

3326 **CK\_GCM\_MESSAGE\_PARAMS\_PTR** is a pointer to a **CK\_GCM\_MESSAGE\_PARAMS**.  
 3327

3328 ♦ **CK\_CCM\_PARAMS; CK\_CCM\_PARAMS\_PTR**

3329 **CK\_CCM\_PARAMS** is a structure that provides the parameters to the **CKM\_AES\_CCM** mechanism  
 3330 when used for Encrypt or Decrypt. It is defined as follows:

```
3331 typedef struct CK_CCM_PARAMS {
3332 CK ULONG ulDataLen; /*plaintext or ciphertext*/
3333 CK_BYTE_PTR pNonce;
3334 CK ULONG ulNonceLen;
3335 CK_BYTE_PTR pAAD;
3336 CK ULONG ulAADLen;
3337 CK ULONG ulMACLen;
3338 } CK_CCM_PARAMS;
```

3339 The fields of the structure have the following meanings, where L is the size in bytes of the data length's  
 3340 length ( $2 \leq L \leq 8$ ):

|      |                   |                                                                                                                 |
|------|-------------------|-----------------------------------------------------------------------------------------------------------------|
| 3341 | <i>ulDataLen</i>  | <i>length of the data where <math>0 \leq ulDataLen &lt; 2^{8L}</math>.</i>                                      |
| 3342 | <i>pNonce</i>     | <i>the nonce.</i>                                                                                               |
| 3343 | <i>ulNonceLen</i> | <i>length of pNonce in bytes where <math>7 \leq ulNonceLen \leq 13</math>.</i>                                  |
| 3344 | <i>pAAD</i>       | <i>Additional authentication data. This data is authenticated but not encrypted.</i>                            |
| 3346 | <i>ulAADLen</i>   | <i>length of pAAD in bytes where <math>0 \leq ulAADLen \leq (2^{32}) - 1</math>.</i>                            |
| 3347 | <i>ulMACLen</i>   | <i>length of the MAC (output following cipher text) in bytes. Valid values are 4, 6, 8, 10, 12, 14, and 16.</i> |

3349 **CK\_CCM\_PARAMS\_PTR** is a pointer to a **CK\_CCM\_PARAMS**.

3350 ♦ **CK\_CCM\_MESSAGE\_PARAMS; CK\_CCM\_MESSAGE\_PARAMS\_PTR**

3351 **CK\_CCM\_MESSAGE\_PARAMS** is a structure that provides the parameters to the **CKM\_AES\_CCM**  
 3352 mechanism when used for MessageEncrypt or MessageDecrypt. It is defined as follows:

```
3353 typedef struct CK_CCM_MESSAGE_PARAMS {
3354 CK ULONG ulDataLen; /*plaintext or ciphertext*/
3355 CK_BYTE_PTR pNonce;
3356 CK ULONG ulNonceLen;
3357 CK ULONG ulNonceFixedBits;
3358 CK_GENERATOR_FUNCTION nonceGenerator;
3359 CK_BYTE_PTR pMAC;
3360 CK ULONG ulMACLen;
3361 } CK_CCM_MESSAGE_PARAMS;
```

3362

3363 The fields of the structure have the following meanings, where L is the size in bytes of the data length's  
 3364 length ( $2 \leq L \leq 8$ ):

|      |                  |                                                                            |
|------|------------------|----------------------------------------------------------------------------|
| 3365 | <i>ulDataLen</i> | <i>length of the data where <math>0 \leq ulDataLen &lt; 2^{8L}</math>.</i> |
|------|------------------|----------------------------------------------------------------------------|

|      |                         |                                                                                                                                                            |
|------|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3366 | <i>pNonce</i>           | <i>the nonce.</i>                                                                                                                                          |
| 3367 | <i>ulNonceLen</i>       | <i>length of pNonce in bytes where 7 &lt;= ulNonceLen &lt;= 13.</i>                                                                                        |
| 3368 | <i>ulNonceFixedBits</i> | <i>number of bits of the original nonce to preserve when generating a new nonce. These bits are counted from the Most significant bits (to the right).</i> |
| 3369 |                         |                                                                                                                                                            |
| 3370 |                         |                                                                                                                                                            |
| 3371 | <i>nonceGenerator</i>   | <i>Function used to generate a new nonce. Each nonce must be unique for a given session.</i>                                                               |
| 3372 |                         |                                                                                                                                                            |
| 3373 | <i>pMAC</i>             | <i>location of the CCM MAC returned on MessageEncrypt, provided on MessageDecrypt</i>                                                                      |
| 3374 |                         |                                                                                                                                                            |
| 3375 | <i>ulMACLen</i>         | <i>length of the MAC (output following cipher text) in bytes. Valid values are 4, 6, 8, 10, 12, 14, and 16.</i>                                            |
| 3376 |                         |                                                                                                                                                            |

3377 **CK\_CCM\_MESSAGE\_PARAMS\_PTR** is a pointer to a **CK\_CCM\_MESSAGE\_PARAMS**.

3378

## 3379 2.14 AES CMAC

3380 *Table 85, Mechanisms vs. Functions*

| Mechanism            | Functions         |               |                      |        |                    |               |        |
|----------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                      | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_AES_CMAC_GENERAL |                   | ✓             |                      |        |                    |               |        |
| CKM_AES_CMAC         |                   | ✓             |                      |        |                    |               |        |

3381 <sup>1</sup> SR = SignRecover, VR = VerifyRecover .

### 3382 2.14.1 Definitions

3383 Mechanisms:

3384 CKM\_AES\_CMAC\_GENERAL  
3385 CKM\_AES\_CMAC

### 3386 2.14.2 Mechanism parameters

3387 CKM\_AES\_CMAC\_GENERAL uses the existing **CK\_MAC\_GENERAL\_PARAMS** structure.  
3388 CKM\_AES\_CMAC does not use a mechanism parameter.

### 3389 2.14.3 General-length AES-CMAC

3390 General-length AES-CMAC, denoted **CKM\_AES\_CMAC\_GENERAL**, is a mechanism for single- and  
3391 multiple-part signatures and verification, based on [NIST SP800-38B] and [RFC 4493].

3392 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS** structure, which specifies the output length  
3393 desired from the mechanism.

3394 The output bytes from this mechanism are taken from the start of the final AES cipher block produced in  
3395 the MACing process.

3396 Constraints on key types and the length of data are summarized in the following table:

3397 *Table 86, General-length AES-CMAC: Key And Data Length*

| <b>Function</b> | <b>Key type</b> | <b>Data length</b> | <b>Signature length</b>                  |
|-----------------|-----------------|--------------------|------------------------------------------|
| C_Sign          | CKK_AES         | any                | 1-block size, as specified in parameters |
| C_Verify        | CKK_AES         | any                | 1-block size, as specified in parameters |

3398 References [NIST SP800-38B] and [RFC 4493] recommend that the output MAC is not truncated to less  
 3399 than 64 bits. The MAC length must be specified before the communication starts, and must not be  
 3400 changed during the lifetime of the key. It is the caller's responsibility to follow these rules.

3401 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 3402 specify the supported range of AES key sizes, in bytes.

#### 3403 **2.14.4 AES-CMAC**

3404 AES-CMAC, denoted **CKM\_AES\_CMAC**, is a special case of the general-length AES-CMAC mechanism.  
 3405 AES-MAC always produces and verifies MACs that are a full block size in length, the default output length  
 3406 specified by [RFC 4493].

3407 Constraints on key types and the length of data are summarized in the following table:

3408 *Table 87, AES-CMAC: Key And Data Length*

| <b>Function</b> | <b>Key type</b> | <b>Data length</b> | <b>Signature length</b> |
|-----------------|-----------------|--------------------|-------------------------|
| C_Sign          | CKK_AES         | any                | Block size (16 bytes)   |
| C_Verify        | CKK_AES         | any                | Block size (16 bytes)   |

3409 References [NIST SP800-38B] and [RFC 4493] recommend that the output MAC is not truncated to less  
 3410 than 64 bits. The MAC length must be specified before the communication starts, and must not be  
 3411 changed during the lifetime of the key. It is the caller's responsibility to follow these rules.

3412 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 3413 specify the supported range of AES key sizes, in bytes.

#### 3414 **2.15 AES XTS**

3415 *Table 88, Mechanisms vs. Functions*

| <b>Mechanism</b>    | <b>Functions</b>             |                          |                                |               |                           |                          |               |
|---------------------|------------------------------|--------------------------|--------------------------------|---------------|---------------------------|--------------------------|---------------|
|                     | <b>Encrypt &amp; Decrypt</b> | <b>Sign &amp; Verify</b> | <b>SR &amp; VR<sup>1</sup></b> | <b>Digest</b> | <b>Gen. Key/ Key Pair</b> | <b>Wrap &amp; Unwrap</b> | <b>Derive</b> |
| CKM_AES_XTS         | ✓                            |                          |                                |               |                           | ✓                        |               |
| CKM_AES_XTS_KEY_GEN |                              |                          |                                |               | ✓                         |                          |               |

#### 3416 **2.15.1 Definitions**

3417 This section defines the key type "CKK\_AES\_XTS" for type CK\_KEY\_TYPE as used in the  
 3418 CKA\_KEY\_TYPE attribute of key objects.

3419 Mechanisms:

3420       CKM\_AES\_XTS

3421       CKM\_AES\_XTS\_KEY\_GEN

3422 **2.15.2 AES-XTS secret key objects**

3423 *Table 89, AES-XTS Secret Key Object Attributes*

| <b>Attribute</b>               | <b>Data type</b> | <b>Meaning</b>               |
|--------------------------------|------------------|------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup>   | Byte array       | Key value (32 or 64 bytes)   |
| CKA_VALUE_LEN <sup>2,3,6</sup> | CK ULONG         | Length in bytes of key value |

3424 - Refer to [PKCS11-Base] table 11 for footnotes

3425 **2.15.3 AES-XTS key generation**

3426 The double-length AES-XTS key generation mechanism, denoted **CKM\_AES\_XTS\_KEY\_GEN**, is a key  
3427 generation mechanism for double-length AES-XTS keys.

3428 The mechanism generates AES-XTS keys with a particular length in bytes as specified in the  
3429 CKA\_VALUE\_LEN attributes of the template for the key.

3430 This mechanism contributes the CKA\_CLASS, CKA\_KEY\_TYPE, and CKA\_VALUE attributes to the new  
3431 key. Other attributes supported by the double-length AES-XTS key type (specifically, the flags indicating  
3432 which functions the key supports) may be specified in the template for the key, or else are assigned  
3433 default initial values.

3434 For this mechanism, the ulMinKeySize and ulMaxKeySize fields of the CK\_MECHANISM\_INFO structure  
3435 specify the supported range of AES-XTS key sizes, in bytes.

3436 **2.15.4 AES-XTS**

3437 AES-XTS (XEX-based Tweaked CodeBook mode with CipherText Stealing), denoted **CKM\_AES\_XTS**,  
3438 is a mechanism for single- and multiple-part encryption and decryption. It is specified in NIST SP800-38E.

3439 Its single parameter is a Data Unit Sequence Number 16 bytes long. Supported key lengths are 32 and  
3440 64 bytes. Keys are internally split into half-length sub-keys of 16 and 32 bytes respectively. Constraints on  
3441 key types and the length of data are summarized in the following table:

3442 *Table 90, AES-XTS: Key And Data Length*

| <b>Function</b> | <b>Key type</b> | <b>Input length</b>          | <b>Output length</b> | <b>Comments</b> |
|-----------------|-----------------|------------------------------|----------------------|-----------------|
| C_Encrypt       | CKK_AES_XTS     | Any, ≥ block size (16 bytes) | Same as input length | No final part   |
| C_Decrypt       | CKK_AES_XTS     | Any, ≥ block size (16 bytes) | Same as input length | No final part   |

3443

3444 **2.16 AES Key Wrap**

3445 *Table 91, AES Key Wrap Mechanisms vs. Functions*

3446

| <b>Mechanism</b>     | <b>Functions</b>             |                          |                                |               |                           |                          |               |
|----------------------|------------------------------|--------------------------|--------------------------------|---------------|---------------------------|--------------------------|---------------|
|                      | <b>Encrypt &amp; Decrypt</b> | <b>Sign &amp; Verify</b> | <b>SR &amp; VR<sup>1</sup></b> | <b>Digest</b> | <b>Gen. Key/ Key Pair</b> | <b>Wrap &amp; Unwrap</b> | <b>Derive</b> |
| CKM_AES_KEY_WRAP     | ✓                            |                          |                                |               |                           | ✓                        |               |
| CKM_AES_KEY_WRAP_PAD | ✓                            |                          |                                |               |                           | ✓                        |               |
| CKM_AES_KEY_WRAP_KWP | ✓                            |                          |                                |               |                           | ✓                        |               |

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

3447 **2.16.1 Definitions**

3448 Mechanisms:

3449 CKM\_AES\_KEY\_WRAP  
3450 CKM\_AES\_KEY\_WRAP\_PAD  
3451 CKM\_AES\_KEY\_WRAP\_KWP

3452 **2.16.2 AES Key Wrap Mechanism parameters**

3453 The mechanisms will accept an optional mechanism parameter as the Initialization vector which, if  
3454 present, must be a fixed size array of 8 bytes for CKM\_AES\_KEY\_WRAP and  
3455 CKM\_AES\_KEY\_WRAP\_PAD, resp. 4 bytes for CKM\_AES\_KEY\_WRAP\_KWP; and, if NULL, will use  
3456 the default initial value defined in Section 4.3 resp. 6.2 / 6.3 of [AES KEYWRAP].

3457 The type of this parameter is CK\_BYTE\_PTR and the pointer points to the array of bytes to be used as  
3458 the initial value. The length shall be either 0 and the pointer NULL; or 8 for CKM\_AES\_KEY\_WRAP /  
3459 CKM\_AES\_KEY\_WRAP\_PAD, resp. 4 for CKM\_AES\_KEY\_WRAP\_KWP, and the pointer non-NUL.

3460 **2.16.3 AES Key Wrap**

3461 The mechanisms support only single-part operations, single part wrapping and unwrapping, and single-  
3462 part encryption and decryption.

3463 The CKM\_AES\_KEY\_WRAP mechanism can only wrap a key resp. encrypt a block of data whose size is  
3464 an exact multiple of the AES Key Wrap algorithm block size. Wrapping / encryption is done as defined in  
3465 Section 6.2 of [AES KEYWRAP].

3466 The CKM\_AES\_KEY\_WRAP\_PAD mechanism can wrap a key or encrypt a block of data of any length. It  
3467 does the padding detailed in PKCS #7 of inputs (keys or data blocks), always producing wrapped output  
3468 that is larger than the input key/data to be wrapped. This padding is done by the token before being  
3469 passed to the AES key wrap algorithm, which then wraps / encrypts the padded block of data as defined  
3470 in Section 6.2 of [AES KEYWRAP].

3471 The CKM\_AES\_KEY\_WRAP\_KWP mechanism can wrap a key or encrypt block of data of any length.  
3472 The input is padded and wrapped / encrypted as defined in Section 6.3 of [AES KEYWRAP], which  
3473 produces same results as RFC 5649.

3474 **2.17 Key derivation by data encryption – DES & AES**

3475 These mechanisms allow derivation of keys using the result of an encryption operation as the key value.  
3476 They are for use with the C\_DeriveKey function.

3477 *Table 92, Key derivation by data encryption Mechanisms vs. Functions*

| Mechanism                 | Functions         |               |                      |        |                    |               |        |
|---------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                           | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_DES_ECB_ENCRYPT_DATA  |                   |               |                      |        |                    |               | ✓      |
| CKM_DES_CBC_ENCRYPT_DATA  |                   |               |                      |        |                    |               | ✓      |
| CKM_DES3_ECB_ENCRYPT_DATA |                   |               |                      |        |                    |               | ✓      |
| CKM_DES3_CBC_ENCRYPT_DATA |                   |               |                      |        |                    |               | ✓      |
| CKM_AES_ECB_ENCRYPT_DATA  |                   |               |                      |        |                    |               | ✓      |
| CKM_AES_CBC_ENCRYPT_DATA  |                   |               |                      |        |                    |               | ✓      |

3478 **2.17.1 Definitions**

3479 Mechanisms:

```
3480 CKM_DES_ECB_ENCRYPT_DATA
3481 CKM_DES_CBC_ENCRYPT_DATA
3482 CKM_DES3_ECB_ENCRYPT_DATA
3483 CKM_DES3_CBC_ENCRYPT_DATA
3484 CKM_AES_ECB_ENCRYPT_DATA
3485 CKM_AES_CBC_ENCRYPT_DATA

3486
3487 typedef struct CK_DES_CBC_ENCRYPT_DATA_PARAMS {
3488 CK_BYTE iv[8];
3489 CK_BYTE_PTR pData;
3490 CK ULONG length;
3491 } CK_DES_CBC_ENCRYPT_DATA_PARAMS;

3492
3493 typedef CK_DES_CBC_ENCRYPT_DATA_PARAMS CK_PTR
3494 CK_DES_CBC_ENCRYPT_DATA_PARAMS_PTR;

3495
3496 typedef struct CK_AES_CBC_ENCRYPT_DATA_PARAMS {
3497 CK_BYTE iv[16];
3498 CK_BYTE_PTR pData;
3499 CK ULONG length;
3500 } CK_AES_CBC_ENCRYPT_DATA_PARAMS;

3501
3502 typedef CK_AES_CBC_ENCRYPT_DATA_PARAMS CK_PTR
3503 CK_AES_CBC_ENCRYPT_DATA_PARAMS_PTR;
```

3504 **2.17.2 Mechanism Parameters**

3505 Uses CK\_KEY\_DERIVATION\_STRING\_DATA as defined in section 2.43.2

3506 *Table 93, Mechanism Parameters*

|                                                       |                                                                                                                                                      |
|-------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKM_DES_ECB_ENCRYPT_DATA<br>CKM_DES3_ECB_ENCRYPT_DATA | Uses CK_KEY_DERIVATION_STRING_DATA structure. Parameter is the data to be encrypted and must be a multiple of 8 bytes long.                          |
| CKM_AES_ECB_ENCRYPT_DATA                              | Uses CK_KEY_DERIVATION_STRING_DATA structure. Parameter is the data to be encrypted and must be a multiple of 16 long.                               |
| CKM_DES_CBC_ENCRYPT_DATA<br>CKM_DES3_CBC_ENCRYPT_DATA | Uses CK_DES_CBC_ENCRYPT_DATA_PARAMS. Parameter is an 8 byte IV value followed by the data. The data value part must be a multiple of 8 bytes long.   |
| CKM_AES_CBC_ENCRYPT_DATA                              | Uses CK_AES_CBC_ENCRYPT_DATA_PARAMS. Parameter is an 16 byte IV value followed by the data. The data value part must be a multiple of 16 bytes long. |

3507 **2.17.3 Mechanism Description**

3508 The mechanisms will function by performing the encryption over the data provided using the base key.  
3509 The resulting cipher text shall be used to create the key value of the resulting key. If not all the cipher text

3510 is used then the part discarded will be from the trailing end (least significant bytes) of the cipher text data.  
 3511 The derived key shall be defined by the attribute template supplied but constrained by the length of cipher  
 3512 text available for the key value and other normal PKCS11 derivation constraints.  
 3513 Attribute template handling, attribute defaulting and key value preparation will operate as per the SHA-1  
 3514 Key Derivation mechanism in section 2.20.5.  
 3515 If the data is too short to make the requested key then the mechanism returns  
 3516 CKR\_DATA\_LEN\_RANGE.

## 3517 2.18 Double and Triple-length DES

3518 *Table 94, Double and Triple-Length DES Mechanisms vs. Functions*

| Mechanism            | Functions         |               |                      |        |                   |               |        |
|----------------------|-------------------|---------------|----------------------|--------|-------------------|---------------|--------|
|                      | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/Key Pair | Wrap & Unwrap | Derive |
| CKM_DES2_KEY_GEN     |                   |               |                      |        | ✓                 |               |        |
| CKM_DES3_KEY_GEN     |                   |               |                      |        | ✓                 |               |        |
| CKM_DES3_ECB         | ✓                 |               |                      |        |                   | ✓             |        |
| CKM_DES3_CBC         | ✓                 |               |                      |        |                   | ✓             |        |
| CKM_DES3_CBC_PAD     | ✓                 |               |                      |        |                   | ✓             |        |
| CKM_DES3_MAC_GENERAL |                   | ✓             |                      |        |                   |               |        |
| CKM_DES3_MAC         |                   | ✓             |                      |        |                   |               |        |

### 3519 2.18.1 Definitions

3520 This section defines the key type “CKK\_DES2” and “CKK\_DES3” for type CK\_KEY\_TYPE as used in the  
 3521 CKA\_KEY\_TYPE attribute of key objects.

3522 Mechanisms:

3523 CKM\_DES2\_KEY\_GEN  
 3524 CKM\_DES3\_KEY\_GEN  
 3525 CKM\_DES3\_ECB  
 3526 CKM\_DES3\_CBC  
 3527 CKM\_DES3\_MAC  
 3528 CKM\_DES3\_MAC\_GENERAL  
 3529 CKM\_DES3\_CBC\_PAD

### 3530 2.18.2 DES2 secret key objects

3531 DES2 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_DES2**) hold double-length  
 3532 DES keys. The following table defines the DES2 secret key object attributes, in addition to the common  
 3533 attributes defined for this object class:

3534 *Table 95, DES2 Secret Key Object Attributes*

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

3535 - Refer to [PKCS11-Base] table 11 for footnotes

3536 DES2 keys must always have their parity bits properly set as described in FIPS PUB 46-3 (*i.e.*, each of  
3537 the DES keys comprising a DES2 key must have its parity bits properly set). Attempting to create or  
3538 unwrap a DES2 key with incorrect parity will return an error.

3539 The following is a sample template for creating a double-length DES secret key object:

```
3540 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
3541 CK_KEY_TYPE keyType = CKK_DES2;
3542 CK_UTF8CHAR label[] = "A DES2 secret key object";
3543 CK_BYTE value[16] = {...};
3544 CK_BBOOL true = CK_TRUE;
3545 CK_ATTRIBUTE template[] = {
3546 {CKA_CLASS, &class, sizeof(class)},
3547 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
3548 {CKA_TOKEN, &true, sizeof(true)},
3549 {CKA_LABEL, label, sizeof(label)-1},
3550 {CKA_ENCRYPT, &true, sizeof(true)},
3551 {CKA_VALUE, value, sizeof(value)}
3552 };
```

3553

3554 CKA\_CHECK\_VALUE: The value of this attribute is derived from the key object by taking the first three  
3555 bytes of the ECB encryption of a single block of null (0x00) bytes, using the default cipher associated with  
3556 the key type of the secret key object.

### 3557 2.18.3 DES3 secret key objects

3558 DES3 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_DES3**) hold triple-length DES  
3559 keys. The following table defines the DES3 secret key object attributes, in addition to the common  
3560 attributes defined for this object class:

3561 *Table 96, DES3 Secret Key Object Attributes*

| Attribute                    | Data type  | Meaning                          |
|------------------------------|------------|----------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup> | Byte array | Key value (always 24 bytes long) |

3562 - Refer to [PKCS11-Base] table 11 for footnotes

3563 DES3 keys must always have their parity bits properly set as described in FIPS PUB 46-3 (*i.e.*, each of  
3564 the DES keys comprising a DES3 key must have its parity bits properly set). Attempting to create or  
3565 unwrap a DES3 key with incorrect parity will return an error.

3566 The following is a sample template for creating a triple-length DES secret key object:

```
3567 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
3568 CK_KEY_TYPE keyType = CKK_DES3;
3569 CK_UTF8CHAR label[] = "A DES3 secret key object";
3570 CK_BYTE value[24] = {...};
3571 CK_BBOOL true = CK_TRUE;
3572 CK_ATTRIBUTE template[] = {
3573 {CKA_CLASS, &class, sizeof(class)},
3574 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
3575 {CKA_TOKEN, &true, sizeof(true)},
3576 {CKA_LABEL, label, sizeof(label)-1},
3577 {CKA_ENCRYPT, &true, sizeof(true)},
3578 {CKA_VALUE, value, sizeof(value)}
3579 };
```

3580  
3581 CKA\_CHECK\_VALUE: The value of this attribute is derived from the key object by taking the first three  
3582 bytes of the ECB encryption of a single block of null (0x00) bytes, using the default cipher associated with  
3583 the key type of the secret key object.

#### 3584 2.18.4 Double-length DES key generation

3585 The double-length DES key generation mechanism, denoted **CKM\_DES2\_KEY\_GEN**, is a key  
3586 generation mechanism for double-length DES keys. The DES keys making up a double-length DES key  
3587 both have their parity bits set properly, as specified in FIPS PUB 46-3.  
3588 It does not have a parameter.

3589 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
3590 key. Other attributes supported by the double-length DES key type (specifically, the flags indicating which  
3591 functions the key supports) may be specified in the template for the key, or else are assigned default  
3592 initial values.

3593 Double-length DES keys can be used with all the same mechanisms as triple-DES keys:  
3594 **CKM\_DES3\_ECB**, **CKM\_DES3\_CBC**, **CKM\_DES3\_CBC\_PAD**, **CKM\_DES3\_MAC\_GENERAL**, and  
3595 **CKM\_DES3\_MAC**. Triple-DES encryption with a double-length DES key is equivalent to encryption with  
3596 a triple-length DES key with K1=K3 as specified in FIPS PUB 46-3.

3597 When double-length DES keys are generated, it is token-dependent whether or not it is possible for either  
3598 of the component DES keys to be “weak” or “semi-weak” keys.

#### 3599 2.18.5 Triple-length DES Order of Operations

3600 Triple-length DES encryptions are carried out as specified in FIPS PUB 46-3: encrypt, decrypt, encrypt.  
3601 Decryptions are carried out with the opposite three steps: decrypt, encrypt, decrypt. The mathematical  
3602 representations of the encrypt and decrypt operations are as follows:

$$3603 \text{DES3-E}(\{K1, K2, K3\}, P) = E(K3, D(K2, E(K1, P))) \\ 3604 \text{DES3-D}(\{K1, K2, K3\}, C) = D(K1, E(K2, D(K3, P)))$$

#### 3605 2.18.6 Triple-length DES in CBC Mode

3606 Triple-length DES operations in CBC mode, with double or triple-length keys, are performed using outer  
3607 CBC as defined in X9.52. X9.52 describes this mode as TCBC. The mathematical representations of the  
3608 CBC encrypt and decrypt operations are as follows:

$$3609 \text{DES3-CBC-E}(\{K1, K2, K3\}, P) = E(K3, D(K2, E(K1, P + I))) \\ 3610 \text{DES3-CBC-D}(\{K1, K2, K3\}, C) = D(K1, E(K2, D(K3, P))) + I$$

3611 The value *I* is either an 8-byte initialization vector or the previous block of cipher text that is added to the  
3612 current input block. The addition operation is used is addition modulo-2 (XOR).

#### 3613 2.18.7 DES and Triple length DES in OFB Mode

3614 Table 97, DES and Triple Length DES in OFB Mode Mechanisms vs. Functions

| Mechanism     | Functions         |               |                      |        |                   |               |        |
|---------------|-------------------|---------------|----------------------|--------|-------------------|---------------|--------|
|               | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/Key Pair | Wrap & Unwrap | Derive |
| CKM_DES_OFB64 | ✓                 |               |                      |        |                   |               |        |
| CKM_DES_OFB8  | ✓                 |               |                      |        |                   |               |        |
| CKM_DES_CFB64 | ✓                 |               |                      |        |                   |               |        |

| Mechanism    | Functions         |               |                      |        |                    |               |        |
|--------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|              | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_DES_CFB8 | ✓                 |               |                      |        |                    |               |        |

3615

3616 Cipher DES has a output feedback mode, DES-OFB, denoted **CKM\_DES\_OFB8** and  
 3617 **CKM\_DES\_OFB64**. It is a mechanism for single and multiple-part encryption and decryption with DES.

3618 It has a parameter, an initialization vector for this mode. The initialization vector has the same length as  
 3619 the block size.

3620 Constraints on key types and the length of data are summarized in the following table:

3621 *Table 98, OFB: Key And Data Length*

| Function  | Key type                          | Input length | Output length        | Comments      |
|-----------|-----------------------------------|--------------|----------------------|---------------|
| C_Encrypt | CKK_DES,<br>CKK_DES2,<br>CKK_DES3 | any          | same as input length | no final part |
| C_Decrypt | CKK_DES,<br>CKK_DES2,<br>CKK_DES3 | any          | same as input length | no final part |

3622 For this mechanism the **CK\_MECHANISM\_INFO** structure is as specified for CBC mode.

## 2.18.8 DES and Triple length DES in CFB Mode

3624 Cipher DES has a cipher feedback mode, DES-CFB, denoted **CKM\_DES\_CFB8** and **CKM\_DES\_CFB64**.  
 3625 It is a mechanism for single and multiple-part encryption and decryption with DES.

3626 It has a parameter, an initialization vector for this mode. The initialization vector has the same length as  
 3627 the block size.

3628 Constraints on key types and the length of data are summarized in the following table:

3629 *Table 99, CFB: Key And Data Length*

| Function  | Key type                          | Input length | Output length        | Comments      |
|-----------|-----------------------------------|--------------|----------------------|---------------|
| C_Encrypt | CKK_DES,<br>CKK_DES2,<br>CKK_DES3 | any          | same as input length | no final part |
| C_Decrypt | CKK_DES,<br>CKK_DES2,<br>CKK_DES3 | any          | same as input length | no final part |

3630 For this mechanism the **CK\_MECHANISM\_INFO** structure is as specified for CBC mode.

## 2.19 Double and Triple-length DES CMAC

3632 *Table 100, Double and Triple-length DES CMAC Mechanisms vs. Functions*

| Mechanism             | Functions         |               |                      |        |                    |               |        |
|-----------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                       | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_DES3_CMAC_GENERAL |                   | ✓             |                      |        |                    |               |        |
| CKM_DES3_CMAC         |                   | ✓             |                      |        |                    |               |        |

3633 1 SR = SignRecover, VR = VerifyRecover.

## 2.19.1 Definitions

3635 Mechanisms:

3636 CKM\_DES3\_CMAC\_GENERAL  
3637 CKM\_DES3\_CMAC

## 2.19.2 Mechanism parameters

3639 CKM\_DES3\_CMAC\_GENERAL uses the existing **CK\_MAC\_GENERAL\_PARAMS** structure.  
3640 CKM\_DES3\_CMAC does not use a mechanism parameter.

## 2.19.3 General-length DES3-MAC

3642 General-length DES3-CMAC, denoted **CKM\_DES3\_CMAC\_GENERAL**, is a mechanism for single- and  
3643 multiple-part signatures and verification with DES3 or DES2 keys, based on [NIST sp800-38b].

3644 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS** structure, which specifies the output length  
3645 desired from the mechanism.

3646 The output bytes from this mechanism are taken from the start of the final DES3 cipher block produced in  
3647 the MACing process.

3648 Constraints on key types and the length of data are summarized in the following table:

3649 *Table 101, General-length DES3-CMAC: Key And Data Length*

| Function | Key type             | Data length | Signature length                         |
|----------|----------------------|-------------|------------------------------------------|
| C_Sign   | CKK_DES3<br>CKK_DES2 | any         | 1-block size, as specified in parameters |
| C_Verify | CKK_DES3<br>CKK_DES2 | any         | 1-block size, as specified in parameters |

3650 Reference [NIST sp800-38b] recommends that the output MAC is not truncated to less than 64 bits  
3651 (which means using the entire block for DES). The MAC length must be specified before the  
3652 communication starts, and must not be changed during the lifetime of the key. It is the caller's  
3653 responsibility to follow these rules.

3654 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
3655 are not used

## 2.19.4 DES3-CMAC

3657 DES3-CMAC, denoted **CKM\_DES3\_CMAC**, is a special case of the general-length DES3-CMAC  
3658 mechanism. DES3-MAC always produces and verifies MACs that are a full block size in length, since the  
3659 DES3 block length is the minimum output length recommended by [NIST sp800-38b].

3660 Constraints on key types and the length of data are summarized in the following table:

3661 *Table 102, DES3-CMAC: Key And Data Length*

| Function | Key type             | Data length | Signature length     |  |
|----------|----------------------|-------------|----------------------|--|
| C_Sign   | CKK_DES3<br>CKK_DES2 | any         | Block size (8 bytes) |  |
| C_Verify | CKK_DES3<br>CKK_DES2 | any         | Block size (8 bytes) |  |

3662 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
3663 are not used.

## 2.20 SHA-1

3665 *Table 103, SHA-1 Mechanisms vs. Functions*

| Mechanism               | Functions         |               |                      |        |                    |               |        |
|-------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                         | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SHA_1               |                   |               |                      | ✓      |                    |               |        |
| CKM_SHA_1_HMAC_GENERAL  |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA_1_HMAC          |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA1_KEY_DERIVATION |                   |               |                      |        |                    |               | ✓      |
| CKM_SHA_1_KEY_GEN       |                   |               |                      |        | ✓                  |               |        |

### 2.20.1 Definitions

3667 This section defines the key type "CKK\_SHA\_1\_HMAC" for type CK\_KEY\_TYPE as used in the  
3668 CKA\_KEY\_TYPE attribute of key objects.

3669 Mechanisms:

3670 CKM\_SHA\_1  
3671 CKM\_SHA\_1\_HMAC  
3672 CKM\_SHA\_1\_HMAC\_GENERAL  
3673 CKM\_SHA1\_KEY\_DERIVATION  
3674 CKM\_SHA\_1\_KEY\_GEN  
3675

### 2.20.2 SHA-1 digest

3677 The SHA-1 mechanism, denoted **CKM\_SHA\_1**, is a mechanism for message digesting, following the  
3678 Secure Hash Algorithm with a 160-bit message digest defined in FIPS PUB 180-2.

3679 It does not have a parameter.

3680 Constraints on the length of input and output data are summarized in the following table. For single-part  
3681 digesting, the data and the digest may begin at the same location in memory.

3682 *Table 104, SHA-1: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 20            |

### 3683 **2.20.3 General-length SHA-1-HMAC**

3684 The general-length SHA-1-HMAC mechanism, denoted **CKM\_SHA\_1\_HMAC\_GENERAL**, is a  
3685 mechanism for signatures and verification. It uses the HMAC construction, based on the SHA-1 hash  
3686 function. The keys it uses are generic secret keys and CKK\_SHA\_1\_HMAC.

3687 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
3688 output. This length should be in the range 1-20 (the output size of SHA-1 is 20 bytes). Signatures  
3689 (MACs) produced by this mechanism will be taken from the start of the full 20-byte HMAC output.

3690 *Table 105, General-length SHA-1-HMAC: Key And Data Length*

| Function | Key type                             | Data length | Signature length              |
|----------|--------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret<br>CKK_SHA_1_<br>HMAC | any         | 1-20, depending on parameters |
| C_Verify | generic secret<br>CKK_SHA_1_<br>HMAC | any         | 1-20, depending on parameters |

### 3691 **2.20.4 SHA-1-HMAC**

3692 The SHA-1-HMAC mechanism, denoted **CKM\_SHA\_1\_HMAC**, is a special case of the general-length  
3693 SHA-1-HMAC mechanism in Section 2.20.3.

3694 It has no parameter, and always produces an output of length 20.

### 3695 **2.20.5 SHA-1 key derivation**

3696 SHA-1 key derivation, denoted **CKM\_SHA1\_KEY\_DERIVATION**, is a mechanism which provides the  
3697 capability of deriving a secret key by digesting the value of another secret key with SHA-1.

3698 The value of the base key is digested once, and the result is used to make the value of derived secret  
3699 key.

- 3700 • If no length or key type is provided in the template, then the key produced by this mechanism will be a  
3701 generic secret key. Its length will be 20 bytes (the output size of SHA-1).
- 3702 • If no key type is provided in the template, but a length is, then the key produced by this mechanism  
3703 will be a generic secret key of the specified length.
- 3704 • If no length was provided in the template, but a key type is, then that key type must have a well-  
3705 defined length. If it does, then the key produced by this mechanism will be of the type specified in the  
3706 template. If it doesn't, an error will be returned.
- 3707 • If both a key type and a length are provided in the template, the length must be compatible with that  
3708 key type. The key produced by this mechanism will be of the specified type and length.

3709 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key will be set  
3710 properly.

3711 If the requested type of key requires more than 20 bytes, such as DES3, an error is generated.

3712 This mechanism has the following rules about key sensitivity and extractability:

- 3713 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
3714 be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
3715 default value.

- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its **CKA\_EXTRACTABLE** attribute.

## 2.20.6 SHA-1 HMAC key generation

The SHA-1-HMAC key generation mechanism, denoted **CKM\_SHA\_1\_KEY\_GEN**, is a key generation mechanism for NIST's SHA-1-HMAC.

It does not have a parameter.

The mechanism generates SHA-1-HMAC 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 SHA-1-HMAC 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 **CKM\_SHA\_1\_HMAC** key sizes, in bytes.

## 2.21 SHA-224

Table 106, SHA-224 Mechanisms vs. Functions

| Mechanism                 | Functions         |               |                      |        |                   |               |        |
|---------------------------|-------------------|---------------|----------------------|--------|-------------------|---------------|--------|
|                           | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/Key Pair | Wrap & Unwrap | Derive |
| CKM_SHA224                |                   |               |                      | ✓      |                   |               |        |
| CKM_SHA224_HMAC           |                   | ✓             |                      |        |                   |               |        |
| CKM_SHA224_HMAC_GENERAL   |                   | ✓             |                      |        |                   |               |        |
| CKM_SHA224_RSA_PKCS       |                   | ✓             |                      |        |                   |               |        |
| CKM_SHA224_RSA_PKCS_PSS   |                   | ✓             |                      |        |                   |               |        |
| CKM_SHA224_KEY_DERIVATION |                   |               |                      |        |                   |               | ✓      |
| CKM_SHA224_KEY_GEN        |                   |               |                      |        | ✓                 |               |        |

## 2.21.1 Definitions

This section defines the key type "CKK\_SHA224\_HMAC" for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.

Mechanisms:

- CKM\_SHA224
- CKM\_SHA224\_HMAC
- CKM\_SHA224\_HMAC\_GENERAL
- CKM\_SHA224\_KEY\_DERIVATION
- CKM\_SHA224\_KEY\_GEN

## 3747 2.21.2 SHA-224 digest

3748 The SHA-224 mechanism, denoted **CKM\_SHA224**, is a mechanism for message digesting, following the  
3749 Secure Hash Algorithm with a 224-bit message digest defined in 0.

3750 It does not have a parameter.

3751 Constraints on the length of input and output data are summarized in the following table. For single-part  
3752 digesting, the data and the digest may begin at the same location in memory.

3753 *Table 107, SHA-224: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 28            |

## 3754 2.21.3 General-length SHA-224-HMAC

3755 The general-length SHA-224-HMAC mechanism, denoted **CKM\_SHA224\_HMAC\_GENERAL**, is the  
3756 same as the general-length SHA-1-HMAC mechanism except that it uses the HMAC construction based  
3757 on the SHA-224 hash function and length of the output should be in the range 1-28. The keys it uses are  
3758 generic secret keys and CKK\_SHA224\_HMAC. FIPS-198 compliant tokens may require the key length to  
3759 be at least 14 bytes; that is, half the size of the SHA-224 hash output.

3760 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
3761 output. This length should be in the range 1-28 (the output size of SHA-224 is 28 bytes). FIPS-198  
3762 compliant tokens may constrain the output length to be at least 4 or 14 (half the maximum length).  
3763 Signatures (MACs) produced by this mechanism will be taken from the start of the full 28-byte HMAC  
3764 output.

3765 *Table 108, General-length SHA-224-HMAC: Key And Data Length*

| Function | Key type                          | Data length | Signature length              |
|----------|-----------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret<br>CKK_SHA224_HMAC | Any         | 1-28, depending on parameters |
| C_Verify | generic secret<br>CKK_SHA224_HMAC | Any         | 1-28, depending on parameters |

## 3766 2.21.4 SHA-224-HMAC

3767 The SHA-224-HMAC mechanism, denoted **CKM\_SHA224\_HMAC**, is a special case of the general-length  
3768 SHA-224-HMAC mechanism.

3769 It has no parameter, and always produces an output of length 28.

## 3770 2.21.5 SHA-224 key derivation

3771 SHA-224 key derivation, denoted **CKM\_SHA224\_KEY\_DERIVATION**, is the same as the SHA-1 key  
3772 derivation mechanism in Section 12.21.5 except that it uses the SHA-224 hash function and the relevant  
3773 length is 28 bytes.

## 3774 2.21.6 SHA-224 HMAC key generation

3775 The SHA-224-HMAC key generation mechanism, denoted **CKM\_SHA224\_KEY\_GEN**, is a key  
3776 generation mechanism for NIST's SHA224-HMAC.

3777 It does not have a parameter.

3778 The mechanism generates SHA224-HMAC keys with a particular length in bytes, as specified in the  
3779 **CKA\_VALUE\_LEN** attribute of the template for the key.

3780 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
3781 key. Other attributes supported by the SHA224-HMAC key type (specifically, the flags indicating which  
3782 functions the key supports) may be specified in the template for the key, or else are assigned default  
3783 initial values.

3784 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
3785 specify the supported range of **CKM\_SHA224\_HMAC** key sizes, in bytes.

## 3786 **2.22 SHA-256**

3787 *Table 109, SHA-256 Mechanisms vs. Functions*

| Mechanism                 | Functions         |               |                      |        |                    |               |        |
|---------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                           | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SHA256                |                   |               |                      | ✓      |                    |               |        |
| CKM_SHA256_HMAC_GENERAL   |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA256_HMAC           |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA256_KEY_DERIVATION |                   |               |                      |        |                    |               | ✓      |
| CKM_SHA256_KEY_GEN        |                   |               |                      |        | ✓                  |               |        |

### 3788 **2.22.1 Definitions**

3789 This section defines the key type “CKK\_SHA256\_HMAC” for type **CK\_KEY\_TYPE** as used in the  
3790 **CKA\_KEY\_TYPE** attribute of key objects.

3791 Mechanisms:

3792 CKM\_SHA256

3793 CKM\_SHA256\_HMAC

3794 CKM\_SHA256\_HMAC\_GENERAL

3795 CKM\_SHA256\_KEY\_DERIVATION

3796 CKM\_SHA256\_KEY\_GEN

### 3797 **2.22.2 SHA-256 digest**

3798 The SHA-256 mechanism, denoted **CKM\_SHA256**, is a mechanism for message digesting, following the  
3799 Secure Hash Algorithm with a 256-bit message digest defined in FIPS PUB 180-2.

3800 It does not have a parameter.

3801 Constraints on the length of input and output data are summarized in the following table. For single-part  
3802 digesting, the data and the digest may begin at the same location in memory.

3803 *Table 110, SHA-256: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 32            |

### 3804 **2.22.3 General-length SHA-256-HMAC**

3805 The general-length SHA-256-HMAC mechanism, denoted **CKM\_SHA256\_HMAC\_GENERAL**, is the  
3806 same as the general-length SHA-1-HMAC mechanism in Section 2.20.3, except that it uses the HMAC  
3807 construction based on the SHA-256 hash function and length of the output should be in the range 1-32.  
3808 The keys it uses are generic secret keys and CKK\_SHA256\_HMAC. FIPS-198 compliant tokens may  
3809 require the key length to be at least 16 bytes; that is, half the size of the SHA-256 hash output.

3810 It has a parameter, a CK\_MAC\_GENERAL\_PARAMS, which holds the length in bytes of the desired  
 3811 output. This length should be in the range 1-32 (the output size of SHA-256 is 32 bytes). FIPS-198  
 3812 compliant tokens may constrain the output length to be at least 4 or 16 (half the maximum length).  
 3813 Signatures (MACs) produced by this mechanism will be taken from the start of the full 32-byte HMAC  
 3814 output.

3815 *Table 111, General-length SHA-256-HMAC: Key And Data Length*

| <b>Function</b> | <b>Key type</b>                        | <b>Data length</b> | <b>Signature length</b>       |
|-----------------|----------------------------------------|--------------------|-------------------------------|
| C_Sign          | generic secret,<br>CKK_SHA256_<br>HMAC | Any                | 1-32, depending on parameters |
| C_Verify        | generic secret,<br>CKK_SHA256_<br>HMAC | Any                | 1-32, depending on parameters |

## 3816 **2.22.4 SHA-256-HMAC**

3817 The SHA-256-HMAC mechanism, denoted **CKM\_SHA256\_HMAC**, is a special case of the general-length  
 3818 SHA-256-HMAC mechanism in Section 2.22.3.

3819 It has no parameter, and always produces an output of length 32.

## 3820 **2.22.5 SHA-256 key derivation**

3821 SHA-256 key derivation, denoted CKM\_SHA256\_KEY\_DERIVATION, is the same as the SHA-1 key  
 3822 derivation mechanism in Section 2.20.5, except that it uses the SHA-256 hash function and the relevant  
 3823 length is 32 bytes.

## 3824 **2.22.6 SHA-256 HMAC key generation**

3825 The SHA-256-HMAC key generation mechanism, denoted **CKM\_SHA256\_KEY\_GEN**, is a key  
 3826 generation mechanism for NIST's SHA256-HMAC.

3827 It does not have a parameter.

3828 The mechanism generates SHA256-HMAC keys with a particular length in bytes, as specified in the  
 3829 **CKA\_VALUE\_LEN** attribute of the template for the key.

3830 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
 3831 key. Other attributes supported by the SHA256-HMAC key type (specifically, the flags indicating which  
 3832 functions the key supports) may be specified in the template for the key, or else are assigned default  
 3833 initial values.

3834 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 3835 specify the supported range of **CKM\_SHA256\_HMAC** key sizes, in bytes.

## 3836 **2.23 SHA-384**

3837 *Table 112, SHA-384 Mechanisms vs. Functions*

| <b>Mechanism</b>        | <b>Functions</b>             |                          |                                |               |                           |                          |               |
|-------------------------|------------------------------|--------------------------|--------------------------------|---------------|---------------------------|--------------------------|---------------|
|                         | <b>Encrypt &amp; Decrypt</b> | <b>Sign &amp; Verify</b> | <b>SR &amp; VR<sup>1</sup></b> | <b>Digest</b> | <b>Gen. Key/ Key Pair</b> | <b>Wrap &amp; Unwrap</b> | <b>Derive</b> |
| CKM_SHA384              |                              |                          |                                | ✓             |                           |                          |               |
| CKM_SHA384_HMAC_GENERAL |                              | ✓                        |                                |               |                           |                          |               |
| CKM_SHA384_HMAC         |                              | ✓                        |                                |               |                           |                          |               |

| Mechanism                 | Functions         |               |                      |        |                    |               |        |
|---------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                           | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SHA384_KEY_DERIVATION |                   |               |                      |        |                    |               | ✓      |
| CKM_SHA384_KEY_GEN        |                   |               |                      |        | ✓                  |               |        |

### 2.23.1 Definitions

This section defines the key type “CKK\_SHA384\_HMAC” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.

CKM\_SHA384  
 CKM\_SHA384\_HMAC  
 CKM\_SHA384\_HMAC\_GENERAL  
 CKM\_SHA384\_KEY\_DERIVATION  
 CKM\_SHA384\_KEY\_GEN

### 2.23.2 SHA-384 digest

The SHA-384 mechanism, denoted **CKM\_SHA384**, is a mechanism for message digesting, following the Secure Hash Algorithm with a 384-bit message digest defined in FIPS PUB 180-2.

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 113, SHA-384: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 48            |

### 2.23.3 General-length SHA-384-HMAC

The general-length SHA-384-HMAC mechanism, denoted **CKM\_SHA384\_HMAC\_GENERAL**, is the same as the general-length SHA-1-HMAC mechanism in Section 2.20.3, except that it uses the HMAC construction based on the SHA-384 hash function and length of the output should be in the range 1-48.

The keys it uses are generic secret keys and CKK\_SHA384\_HMAC. FIPS-198 compliant tokens may require the key length to be at least 24 bytes; that is, half the size of the SHA-384 hash output.

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 0-48 (the output size of SHA-384 is 48 bytes). FIPS-198 compliant tokens may constrain the output length to be at least 4 or 24 (half the maximum length). Signatures (MACs) produced by this mechanism will be taken from the start of the full 48-byte HMAC output.

3864 Table 114, General-length SHA-384-HMAC: Key And Data Length

| Function | Key type                           | Data length | Signature length              |
|----------|------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret,<br>CKK_SHA384_HMAC | Any         | 1-48, depending on parameters |
| C_Verify | generic secret,<br>CKK_SHA384_HMAC | Any         | 1-48, depending on parameters |

3865

3866 

## 2.23.4 SHA-384-HMAC

3867 The SHA-384-HMAC mechanism, denoted **CKM\_SHA384\_HMAC**, is a special case of the general-length  
3868 SHA-384-HMAC mechanism.

3869 It has no parameter, and always produces an output of length 48.

3870 

## 2.23.5 SHA-384 key derivation

3871 SHA-384 key derivation, denoted **CKM\_SHA384\_KEY\_DERIVATION**, is the same as the SHA-1 key  
3872 derivation mechanism in Section 2.20.5, except that it uses the SHA-384 hash function and the relevant  
3873 length is 48 bytes.3874 

## 2.23.6 SHA-384 HMAC key generation

3875 The SHA-384-HMAC key generation mechanism, denoted **CKM\_SHA384\_KEY\_GEN**, is a key  
3876 generation mechanism for NIST's SHA384-HMAC.

3877 It does not have a parameter.

3878 The mechanism generates SHA384-HMAC keys with a particular length in bytes, as specified in the  
3879 **CKA\_VALUE\_LEN** attribute of the template for the key.3880 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
3881 key. Other attributes supported by the SHA384-HMAC key type (specifically, the flags indicating which  
3882 functions the key supports) may be specified in the template for the key, or else are assigned default  
3883 initial values.3884 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
3885 specify the supported range of **CKM\_SHA384\_HMAC** key sizes, in bytes.3886 

## 2.24 SHA-512

3887 Table 115, SHA-512 Mechanisms vs. Functions

| Mechanism                 | Functions         |               |                      |        |                    |               |        |
|---------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                           | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SHA512                |                   |               |                      | ✓      |                    |               |        |
| CKM_SHA512_HMAC_GENERAL   |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA512_HMAC           |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA512_KEY_DERIVATION |                   |               |                      |        |                    |               | ✓      |
| CKM_SHA512_KEY_GEN        |                   |               |                      |        | ✓                  |               |        |

### 3888 2.24.1 Definitions

3889 This section defines the key type "CKK\_SHA512\_HMAC" for type CK\_KEY\_TYPE as used in the  
3890 CKA\_KEY\_TYPE attribute of key objects.

3891 Mechanisms:

3892 CKM\_SHA512  
3893 CKM\_SHA512\_HMAC  
3894 CKM\_SHA512\_HMAC\_GENERAL  
3895 CKM\_SHA512\_KEY\_DERIVATION  
3896 CKM\_SHA512\_KEY\_GEN

### 3897 2.24.2 SHA-512 digest

3898 The SHA-512 mechanism, denoted **CKM\_SHA512**, is a mechanism for message digesting, following the  
3899 Secure Hash Algorithm with a 512-bit message digest defined in FIPS PUB 180-2.

3900 It does not have a parameter.

3901 Constraints on the length of input and output data are summarized in the following table. For single-part  
3902 digesting, the data and the digest may begin at the same location in memory.

3903 *Table 116, SHA-512: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 64            |

### 3904 2.24.3 General-length SHA-512-HMAC

3905 The general-length SHA-512-HMAC mechanism, denoted **CKM\_SHA512\_HMAC\_GENERAL**, is the  
3906 same as the general-length SHA-1-HMAC mechanism in Section 2.20.3, except that it uses the HMAC  
3907 construction based on the SHA-512 hash function and length of the output should be in the range 1-64.

3908 The keys it uses are generic secret keys and CKK\_SHA512\_HMAC. FIPS-198 compliant tokens may  
3909 require the key length to be at least 32 bytes; that is, half the size of the SHA-512 hash output.

3910 It has a parameter, a CK\_MAC\_GENERAL\_PARAMS, which holds the length in bytes of the desired  
3911 output. This length should be in the range 0-64 (the output size of SHA-512 is 64 bytes). FIPS-198  
3912 compliant tokens may constrain the output length to be at least 4 or 32 (half the maximum length).  
3913 Signatures (MACs) produced by this mechanism will be taken from the start of the full 64-byte HMAC  
3914 output.

3915 *Table 117, General-length SHA-384-HMAC: Key And Data Length*

| Function | Key type                               | Data length | Signature length              |
|----------|----------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret,<br>CKK_SHA512_<br>HMAC | Any         | 1-64, depending on parameters |
| C_Verify | generic secret,<br>CKK_SHA512_<br>HMAC | Any         | 1-64, depending on parameters |

3916

### 3917 2.24.4 SHA-512-HMAC

3918 The SHA-512-HMAC mechanism, denoted **CKM\_SHA512\_HMAC**, is a special case of the general-length  
3919 SHA-512-HMAC mechanism.

3920 It has no parameter, and always produces an output of length 64.

## 2.24.5 SHA-512 key derivation

SHA-512 key derivation, denoted **CKM\_SHA512\_KEY\_DERIVATION**, is the same as the SHA-1 key derivation mechanism in Section 2.20.5, except that it uses the SHA-512 hash function and the relevant length is 64 bytes.

## 2.24.6 SHA-512 HMAC key generation

The SHA-512-HMAC key generation mechanism, denoted **CKM\_SHA512\_KEY\_GEN**, is a key generation mechanism for NIST's SHA512-HMAC.

It does not have a parameter.

The mechanism generates SHA512-HMAC 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 SHA512-HMAC 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 **CKM\_SHA512\_HMAC** key sizes, in bytes.

## 2.25 SHA-512/224

Table 118, SHA-512/224 Mechanisms vs. Functions

| Mechanism                     | Functions         |               |                      |        |                   |               |        |
|-------------------------------|-------------------|---------------|----------------------|--------|-------------------|---------------|--------|
|                               | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/Key Pair | Wrap & Unwrap | Derive |
| CKM_SHA512_224                |                   |               |                      | ✓      |                   |               |        |
| CKM_SHA512_224_HMAC_GENERAL   |                   | ✓             |                      |        |                   |               |        |
| CKM_SHA512_224_HMAC           |                   | ✓             |                      |        |                   |               |        |
| CKM_SHA512_224_KEY_DERIVATION |                   |               |                      |        |                   |               | ✓      |
| CKM_SHA512_224_KEY_GEN        |                   |               |                      |        | ✓                 |               |        |

## 2.25.1 Definitions

This section defines the key type "CKK\_SHA512\_224\_HMAC" for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of key objects.

Mechanisms:

CKM\_SHA512\_224

CKM\_SHA512\_224\_HMAC

CKM\_SHA512\_224\_HMAC\_GENERAL

CKM\_SHA512\_224\_KEY\_DERIVATION

CKM\_SHA512\_224\_KEY\_GEN

## 2.25.2 SHA-512/224 digest

The SHA-512/224 mechanism, denoted **CKM\_SHA512\_224**, is a mechanism for message digesting, following the Secure Hash Algorithm defined in FIPS PUB 180-4, section 5.3.6. It is based on a 512-bit

3951 message digest with a distinct initial hash value and truncated to 224 bits. **CKM\_SHA512\_224** is the  
 3952 same as **CKM\_SHA512\_T** with a parameter value of 224.  
 3953 It does not have a parameter.  
 3954 Constraints on the length of input and output data are summarized in the following table. For single-part  
 3955 digesting, the data and the digest may begin at the same location in memory.  
 3956 *Table 119, SHA-512/224: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 28            |

3957 **2.25.3 General-length SHA-512/224-HMAC**  
 3958 The general-length SHA-512/224-HMAC mechanism, denoted **CKM\_SHA512\_224\_HMAC\_GENERAL**,  
 3959 is the same as the general-length SHA-1-HMAC mechanism in Section 2.20.3, except that it uses the  
 3960 HMAC construction based on the SHA-512/224 hash function and length of the output should be in the  
 3961 range 1-28. The keys it uses are generic secret keys and CKK\_SHA512\_224\_HMAC. FIPS-198  
 3962 compliant tokens may require the key length to be at least 14 bytes; that is, half the size of the SHA-  
 3963 512/224 hash output.  
 3964 It has a parameter, a CK\_MAC\_GENERAL\_PARAMS, which holds the length in bytes of the desired  
 3965 output. This length should be in the range 0-28 (the output size of SHA-512/224 is 28 bytes). FIPS-198  
 3966 compliant tokens may constrain the output length to be at least 4 or 14 (half the maximum length).  
 3967 Signatures (MACs) produced by this mechanism will be taken from the start of the full 28-byte HMAC  
 3968 output.

3969 *Table 120, General-length SHA-384-HMAC: Key And Data Length*

| Function | Key type                               | Data length | Signature length              |
|----------|----------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret,<br>CKK_SHA512_224_HMAC | Any         | 1-28, depending on parameters |
| C_Verify | generic secret,<br>CKK_SHA512_224_HMAC | Any         | 1-28, depending on parameters |

3970  
 3971 **2.25.4 SHA-512/224-HMAC**  
 3972 The SHA-512-HMAC mechanism, denoted **CKM\_SHA512\_224\_HMAC**, is a special case of the general-  
 3973 length SHA-512/224-HMAC mechanism.  
 3974 It has no parameter, and always produces an output of length 28.

## 3975 **2.25.5 SHA-512/224 key derivation**

3976 The SHA-512/224 key derivation, denoted **CKM\_SHA512\_224\_KEY\_DERIVATION**, is the same as the  
 3977 SHA-512 key derivation mechanism in section 2.25.5, except that it uses the SHA-512/224 hash function  
 3978 and the relevant length is 28 bytes.

## 3979 **2.25.6 SHA-512/224 HMAC key generation**

3980 The SHA-512/224-HMAC key generation mechanism, denoted **CKM\_SHA512\_224\_KEY\_GEN**, is a key  
 3981 generation mechanism for NIST's SHA512/224-HMAC.  
 3982 It does not have a parameter.  
 3983 The mechanism generates SHA512/224-HMAC keys with a particular length in bytes, as specified in the  
 3984 **CKA\_VALUE\_LEN** attribute of the template for the key.

3985 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
3986 key. Other attributes supported by the SHA512/224-HMAC key type (specifically, the flags indicating  
3987 which functions the key supports) may be specified in the template for the key, or else are assigned  
3988 default initial values.

3989 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
3990 specify the supported range of **CKM\_SHA512\_224\_HMAC** key sizes, in bytes.

## 3991 **2.26 SHA-512/256**

3992 *Table 121, SHA-512/256 Mechanisms vs. Functions*

| Mechanism                     | Functions         |               |                      |        |                   |               |        |
|-------------------------------|-------------------|---------------|----------------------|--------|-------------------|---------------|--------|
|                               | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/Key Pair | Wrap & Unwrap | Derive |
| CKM_SHA512_256                |                   |               |                      | ✓      |                   |               |        |
| CKM_SHA512_256_HMAC_GENERAL   |                   | ✓             |                      |        |                   |               |        |
| CKM_SHA512_256_HMAC           |                   | ✓             |                      |        |                   |               |        |
| CKM_SHA512_256_KEY_DERIVATION |                   |               |                      |        |                   |               | ✓      |
| CKM_SHA512_256_KEY_GEN        |                   |               |                      |        | ✓                 |               |        |

### 3993 **2.26.1 Definitions**

3994 This section defines the key type “CKK\_SHA512\_256\_HMAC” for type CK\_KEY\_TYPE as used in the  
3995 CKA\_KEY\_TYPE attribute of key objects.

3996 Mechanisms:

3997 CKM\_SHA512\_256  
3998 CKM\_SHA512\_256\_HMAC  
3999 CKM\_SHA512\_256\_HMAC\_GENERAL  
4000 CKM\_SHA512\_256\_KEY\_DERIVATION  
4001 CKM\_SHA512\_256\_KEY\_GEN

### 4002 **2.26.2 SHA-512/256 digest**

4003 The SHA-512/256 mechanism, denoted **CKM\_SHA512\_256**, is a mechanism for message digesting,  
4004 following the Secure Hash Algorithm defined in FIPS PUB 180-4, section 5.3.6. It is based on a 512-bit  
4005 message digest with a distinct initial hash value and truncated to 256 bits. **CKM\_SHA512\_256** is the  
4006 same as **CKM\_SHA512\_T** with a parameter value of 256.

4007 It does not have a parameter.

4008 Constraints on the length of input and output data are summarized in the following table. For single-part  
4009 digesting, the data and the digest may begin at the same location in memory.

4010    Table 122, SHA-512/256: Data Length

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 32            |

### 4011    2.26.3 General-length SHA-512/256-HMAC

4012    The general-length SHA-512/256-HMAC mechanism, denoted **CKM\_SHA512\_256\_HMAC\_GENERAL**,  
4013    is the same as the general-length SHA-1-HMAC mechanism in Section 2.20.3, except that it uses the  
4014    HMAC construction based on the SHA-512/256 hash function and length of the output should be in the  
4015    range 1-32. The keys it uses are generic secret keys and CKK\_SHA512\_256\_HMAC. FIPS-198  
4016    compliant tokens may require the key length to be at least 16 bytes; that is, half the size of the SHA-  
4017    512/256 hash output.

4018    It has a parameter, a CK\_MAC\_GENERAL\_PARAMS, which holds the length in bytes of the desired  
4019    output. This length should be in the range 1-32 (the output size of SHA-512/256 is 32 bytes). FIPS-198  
4020    compliant tokens may constrain the output length to be at least 4 or 16 (half the maximum length).  
4021    Signatures (MACs) produced by this mechanism will be taken from the start of the full 32-byte HMAC  
4022    output.

4023    Table 123, General-length SHA-384-HMAC: Key And Data Length

| Function | Key type                               | Data length | Signature length              |
|----------|----------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret,<br>CKK_SHA512_256_HMAC | Any         | 1-32, depending on parameters |
| C_Verify | generic secret,<br>CKK_SHA512_256_HMAC | Any         | 1-32, depending on parameters |

4024

### 4025    2.26.4 SHA-512/256-HMAC

4026    The SHA-512-HMAC mechanism, denoted **CKM\_SHA512\_256\_HMAC**, is a special case of the general-  
4027    length SHA-512/256-HMAC mechanism.

4028    It has no parameter, and always produces an output of length 32.

### 4029    2.26.5 SHA-512/256 key derivation

4030    The SHA-512/256 key derivation, denoted **CKM\_SHA512\_256\_KEY\_DERIVATION**, is the same as the  
4031    SHA-512 key derivation mechanism in section 2.25.5, except that it uses the SHA-512/256 hash function  
4032    and the relevant length is 32 bytes.

### 4033    2.26.6 SHA-512/256 HMAC key generation

4034    The SHA-512/256-HMAC key generation mechanism, denoted **CKM\_SHA512\_256\_KEY\_GEN**, is a key  
4035    generation mechanism for NIST's SHA512/256-HMAC.

4036    It does not have a parameter.

4037    The mechanism generates SHA512/256-HMAC keys with a particular length in bytes, as specified in the  
4038    **CKA\_VALUE\_LEN** attribute of the template for the key.

4039    The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
4040    key. Other attributes supported by the SHA512/256-HMAC key type (specifically, the flags indicating  
4041    which functions the key supports) may be specified in the template for the key, or else are assigned  
4042    default initial values.

4043    For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
4044    specify the supported range of **CKM\_SHA512\_256\_HMAC** key sizes, in bytes.

4045 **2.27 SHA-512/t**

4046 *Table 124, SHA-512 / t Mechanisms vs. Functions*

| Mechanism                   | Functions         |               |                      |        |                    |               |        |
|-----------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                             | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SHA512_T                |                   |               |                      | ✓      |                    |               |        |
| CKM_SHA512_T_HMAC_GENERAL   |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA512_T_HMAC           |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA512_T_KEY_DERIVATION |                   |               |                      |        |                    |               | ✓      |
| CKM_SHA512_T_KEY_GEN        |                   |               |                      |        | ✓                  |               |        |

4047 **2.27.1 Definitions**

4048 This section defines the key type "CKK\_SHA512\_T\_HMAC" for type CK\_KEY\_TYPE as used in the  
 4049 CKA\_KEY\_TYPE attribute of key objects.

4050 Mechanisms:

- 4051 CKM\_SHA512\_T
- 4052 CKM\_SHA512\_T\_HMAC
- 4053 CKM\_SHA512\_T\_HMAC\_GENERAL
- 4054 CKM\_SHA512\_T\_KEY\_DERIVATION
- 4055 CKM\_SHA512\_T\_KEY\_GEN

4056 **2.27.2 SHA-512/t digest**

4057 The SHA-512/t mechanism, denoted **CKM\_SHA512\_T**, is a mechanism for message digesting, following  
 4058 the Secure Hash Algorithm defined in FIPS PUB 180-4, section 5.3.6. It is based on a 512-bit message  
 4059 digest with a distinct initial hash value and truncated to t bits.

4060 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the value of t in bits. The length in  
 4061 bytes of the desired output should be in the range of 0 – ⌈t/8⌉, where 0 < t < 512, and t > 384.

4062 Constraints on the length of input and output data are summarized in the following table. For single-part  
 4063 digesting, the data and the digest may begin at the same location in memory.

4064 *Table 125, SHA-512/256: Data Length*

| Function | Input length | Digest length                         |
|----------|--------------|---------------------------------------|
| C_Digest | any          | ⌈t/8⌉, where 0 < t < 512, and t > 384 |

4065 **2.27.3 General-length SHA-512/t-HMAC**

4066 The general-length SHA-512/t-HMAC mechanism, denoted **CKM\_SHA512\_T\_HMAC\_GENERAL**, is the  
 4067 same as the general-length SHA-1-HMAC mechanism in Section 2.20.3, except that it uses the HMAC  
 4068 construction based on the SHA-512/t hash function and length of the output should be in the range 0 – ⌈  
 4069 t/8⌉, where 0 < t < 512, and t > 384.

## 4070 **2.27.4 SHA-512/t-HMAC**

4071 The SHA-512/t-HMAC mechanism, denoted **CKM\_SHA512\_T\_HMAC**, is a special case of the general-  
4072 length SHA-512/t-HMAC mechanism.

4073 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the value of t in bits. The length in  
4074 bytes of the desired output should be in the range of  $\lceil t/8 \rceil$ , where  $0 < t < 512$ , and  $t \geq 384$ .

## 4075 **2.27.5 SHA-512/t key derivation**

4076 The SHA-512/t key derivation, denoted **CKM\_SHA512\_T\_KEY\_DERIVATION**, is the same as the SHA-  
4077 512 key derivation mechanism in section 2.25.5, except that it uses the SHA-512/t hash function and the  
4078 relevant length is  $\lceil t/8 \rceil$  bytes, where  $0 < t < 512$ , and  $t \geq 384$ .

## 4079 **2.27.6 SHA-512/t HMAC key generation**

4080 The SHA-512/t-HMAC key generation mechanism, denoted **CKM\_SHA512\_T\_KEY\_GEN**, is a key  
4081 generation mechanism for NIST's SHA512/t-HMAC.

4082 It does not have a parameter.

4083 The mechanism generates SHA512/t-HMAC keys with a particular length in bytes, as specified in the  
4084 **CKA\_VALUE\_LEN** attribute of the template for the key.

4085 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
4086 key. Other attributes supported by the SHA512/t-HMAC key type (specifically, the flags indicating which  
4087 functions the key supports) may be specified in the template for the key, or else are assigned default  
4088 initial values.

4089 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
4090 specify the supported range of **CKM\_SHA512\_T\_HMAC** key sizes, in bytes.

4091

## 4092 **2.28 SHA3-224**

4093 *Table 126, SHA-224 Mechanisms vs. Functions*

| Mechanism                   | Functions         |               |                      |        |                    |               |        |
|-----------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                             | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SHA3_224                |                   |               |                      | ✓      |                    |               |        |
| CKM_SHA3_224_HMAC           |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA3_224_HMAC_GENERAL   |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA3_224_KEY_DERIVATION |                   |               |                      |        |                    |               | ✓      |
| CKM_SHA3_224_KEY_GEN        |                   |               |                      |        | ✓                  |               |        |

## 4094 **2.28.1 Definitions**

4095 Mechanisms:

4096 CKM\_SHA3\_224

4097 CKM\_SHA3\_224\_HMAC

4098 CKM\_SHA3\_224\_HMAC\_GENERAL

4099 CKM\_SHA3\_224\_KEY\_DERIVATION

4100 CKM\_SHA3\_224\_KEY\_GEN  
4101  
4102 CKK\_SHA3\_224\_HMAC

## 4103 2.28.2 SHA3-224 digest

4104 The SHA3-224 mechanism, denoted **CKM\_SHA3\_224**, is a mechanism for message digesting, following  
4105 the Secure Hash 3 Algorithm with a 224-bit message digest defined in FIPS Pub 202.

4106 It does not have a parameter.

4107 Constraints on the length of input and output data are summarized in the following table. For single-part  
4108 digesting, the data and the digest may begin at the same location in memory.

4109 *Table 127, SHA3-224: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 28            |

## 4110 2.28.3 General-length SHA3-224-HMAC

4111 The general-length SHA3-224-HMAC mechanism, denoted **CKM\_SHA3\_224\_HMAC\_GENERAL**, is the  
4112 same as the general-length SHA-1-HMAC mechanism in section 2.20.4 except that it uses the HMAC  
4113 construction based on the SHA3-224 hash function and length of the output should be in the range 1-28.  
4114 The keys it uses are generic secret keys and CKK\_SHA3\_224\_HMAC. FIPS-198 compliant tokens may  
4115 require the key length to be at least 14 bytes; that is, half the size of the SHA3-224 hash output.

4116 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
4117 output. This length should be in the range 1-28 (the output size of SHA3-224 is 28 bytes). FIPS-198  
4118 compliant tokens may constrain the output length to be at least 4 or 14 (half the maximum length).  
4119 Signatures (MACs) produced by this mechanism shall be taken from the start of the full 28-byte HMAC  
4120 output.

4121 *Table 128, General-length SHA3-224-HMAC: Key And Data Length*

| Function | Key type                               | Data length | Signature length              |
|----------|----------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret or<br>CKK_SHA3_224_HMAC | Any         | 1-28, depending on parameters |
| C_Verify | generic secret or<br>CKK_SHA3_224_HMAC | Any         | 1-28, depending on parameters |

## 4122 2.28.4 SHA3-224-HMAC

4123 The SHA3-224-HMAC mechanism, denoted **CKM\_SHA3\_224\_HMAC**, is a special case of the general-  
4124 length SHA3-224-HMAC mechanism.

4125 It has no parameter, and always produces an output of length 28.

## 4126 2.28.5 SHA3-224 key derivation

4127 SHA-224 key derivation, denoted **CKM\_SHA3\_224\_KEY\_DERIVATION**, is the same as the SHA-1 key  
4128 derivation mechanism in Section 2.20.5 except that it uses the SHA3-224 hash function and the relevant  
4129 length is 28 bytes.

## 4130 2.28.6 SHA3-224 HMAC key generation

4131 The SHA3-224-HMAC key generation mechanism, denoted **CKM\_SHA3\_224\_KEY\_GEN**, is a key  
4132 generation mechanism for NIST's SHA3-224-HMAC.

4133 It does not have a parameter.

4134 The mechanism generates SHA3-224-HMAC keys with a particular length in bytes, as specified in the  
 4135 **CKA\_VALUE\_LEN** attribute of the template for the key.

4136 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
 4137 key. Other attributes supported by the SHA3-224-HMAC key type (specifically, the flags indicating which  
 4138 functions the key supports) may be specified in the template for the key, or else are assigned default  
 4139 initial values.

4140 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 4141 specify the supported range of **CKM\_SHA3\_224\_HMAC** key sizes, in bytes.

## 4142 **2.29 SHA3-256**

4143 *Table 129, SHA3-256 Mechanisms vs. Functions*

| Mechanism                   | Functions         |               |                      |        |                    |               |        |
|-----------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                             | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SHA3_256                |                   |               |                      | ✓      |                    |               |        |
| CKM_SHA3_256_HMAC_GENERAL   |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA3_256_HMAC           |                   | ✓             |                      |        |                    |               |        |
| CKM_SHA3_256_KEY_DERIVATION |                   |               |                      |        |                    |               | ✓      |
| CKM_SHA3_256_KEY_GEN        |                   |               |                      |        | ✓                  |               |        |

### 4144 **2.29.1 Definitions**

4145 Mechanisms:

4146 CKM\_SHA3\_256

4147 CKM\_SHA3\_256\_HMAC

4148 CKM\_SHA3\_256\_HMAC\_GENERAL

4149 CKM\_SHA3\_256\_KEY\_DERIVATION

4150 CKM\_SHA3\_256\_KEY\_GEN

4151

4152 CKK\_SHA3\_256\_HMAC

### 4153 **2.29.2 SHA3-256 digest**

4154 The SHA3-256 mechanism, denoted **CKM\_SHA3\_256**, is a mechanism for message digesting, following  
 4155 the Secure Hash 3 Algorithm with a 256-bit message digest defined in FIPS PUB 202.

4156 It does not have a parameter.

4157 Constraints on the length of input and output data are summarized in the following table. For single-part  
 4158 digesting, the data and the digest may begin at the same location in memory.

4159 Table 130, SHA3-256: Data Length

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 32            |

## 4160 2.29.3 General-length SHA3-256-HMAC

4161 The general-length SHA3-256-HMAC mechanism, denoted **CKM\_SHA3\_256\_HMAC\_GENERAL**, is the  
4162 same as the general-length SHA-1-HMAC mechanism in Section 2.20.4, except that it uses the HMAC  
4163 construction based on the SHA3-256 hash function and length of the output should be in the range 1-32.  
4164 The keys it uses are generic secret keys and CKK\_SHA3\_256\_HMAC. FIPS-198 compliant tokens may  
4165 require the key length to be at least 16 bytes; that is, half the size of the SHA3-256 hash output.

4166 It has a parameter, a CK\_MAC\_GENERAL\_PARAMS, which holds the length in bytes of the desired  
4167 output. This length should be in the range 1-32 (the output size of SHA3-256 is 32 bytes). FIPS-198  
4168 compliant tokens may constrain the output length to be at least 4 or 16 (half the maximum length).  
4169 Signatures (MACs) produced by this mechanism shall be taken from the start of the full 32-byte HMAC  
4170 output.

4171 Table 131, General-length SHA3-256-HMAC: Key And Data Length

| Function | Key type                               | Data length | Signature length              |
|----------|----------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret or<br>CKK_SHA3_256_HMAC | Any         | 1-32, depending on parameters |
| C_Verify | generic secret or<br>CKK_SHA3_256_HMAC | Any         | 1-32, depending on parameters |

## 4172 2.29.4 SHA3-256-HMAC

4173 The SHA-256-HMAC mechanism, denoted **CKM\_SHA3\_256\_HMAC**, is a special case of the general-  
4174 length SHA-256-HMAC mechanism in Section 2.22.3.

4175 It has no parameter, and always produces an output of length 32.

## 4176 2.29.5 SHA3-256 key derivation

4177 SHA-256 key derivation, denoted CKM\_SHA3\_256\_KEY\_DERIVATION, is the same as the SHA-1 key  
4178 derivation mechanism in Section 2.20.5, except that it uses the SHA3-256 hash function and the relevant  
4179 length is 32 bytes.

## 4180 2.29.6 SHA3-256 HMAC key generation

4181 The SHA3-256-HMAC key generation mechanism, denoted **CKM\_SHA3\_256\_KEY\_GEN**, is a key  
4182 generation mechanism for NIST's SHA3-256-HMAC.

4183 It does not have a parameter.

4184 The mechanism generates SHA3-256-HMAC keys with a particular length in bytes, as specified in the  
4185 **CKA\_VALUE\_LEN** attribute of the template for the key.

4186 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
4187 key. Other attributes supported by the SHA3-256-HMAC key type (specifically, the flags indicating which  
4188 functions the key supports) may be specified in the template for the key, or else are assigned default  
4189 initial values.

4190 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
4191 specify the supported range of **CKM\_SHA3\_256\_HMAC** key sizes, in bytes.

4192

4193 **2.30 SHA3-384**

4194 *Table 132, SHA3-384 Mechanisms vs. Functions*

| Mechanism                   | Functions         |                |                      |        |                    |               |        |
|-----------------------------|-------------------|----------------|----------------------|--------|--------------------|---------------|--------|
|                             | Encrypt & Decrypt | Sign & Verif y | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SHA3_384                |                   | ✓              |                      | ✓      |                    |               |        |
| CKM_SHA3_384_HMAC_GENERAL   |                   | ✓              |                      |        |                    |               |        |
| CKM_SHA3_384_HMAC           |                   | ✓              |                      |        |                    |               |        |
| CKM_SHA3_384_KEY_DERIVATION |                   |                |                      |        |                    |               | ✓      |
| CKM_SHA3_384_KEY_GEN        |                   |                |                      | ✓      |                    |               |        |

4195 **2.30.1 Definitions**

4196 CKM\_SHA3\_384

4197 CKM\_SHA3\_384\_HMAC

4198 CKM\_SHA3\_384\_HMAC\_GENERAL

4199 CKM\_SHA3\_384\_KEY\_DERIVATION

4200 CKM\_SHA3\_384\_KEY\_GEN

4201

4202 CKK\_SHA3\_384\_HMAC

4203 **2.30.2 SHA3-384 digest**

4204 The SHA3-384 mechanism, denoted **CKM\_SHA3\_384**, is a mechanism for message digesting, following  
4205 the Secure Hash 3 Algorithm with a 384-bit message digest defined in FIPS PUB 202.

4206 It does not have a parameter.

4207 Constraints on the length of input and output data are summarized in the following table. For single-part  
4208 digesting, the data and the digest may begin at the same location in memory.

4209 *Table 133, SHA3-384: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 48            |

4210 **2.30.3 General-length SHA3-384-HMAC**

4211 The general-length SHA3-384-HMAC mechanism, denoted **CKM\_SHA3\_384\_HMAC\_GENERAL**, is the  
4212 same as the general-length SHA-1-HMAC mechanism in Section 2.20.4, except that it uses the HMAC  
4213 construction based on the SHA-384 hash function and length of the output should be in the range 1-  
4214 48. The keys it uses are generic secret keys and CKK\_SHA3\_384\_HMAC. FIPS-198 compliant tokens  
4215 may require the key length to be at least 24 bytes; that is, half the size of the SHA3-384 hash output.

4216

4217 It has a parameter, a CK\_MAC\_GENERAL\_PARAMS, which holds the length in bytes of the desired  
4218 output. This length should be in the range 1-48 (the output size of SHA3-384 is 48 bytes). FIPS-198  
4219 compliant tokens may constrain the output length to be at least 4 or 24 (half the maximum length).  
4220 Signatures (MACs) produced by this mechanism shall be taken from the start of the full 48-byte HMAC  
4221 output.

4222 *Table 134, General-length SHA3-384-HMAC: Key And Data Length*

| <b>Function</b> | <b>Key type</b>                     | <b>Data length</b> | <b>Signature length</b>       |
|-----------------|-------------------------------------|--------------------|-------------------------------|
| C_Sign          | generic secret or CKK_SHA3_384_HMAC | Any                | 1-48, depending on parameters |
| C_Verify        | generic secret or CKK_SHA3_384_HMAC | Any                | 1-48, depending on parameters |

4223

## 4224 **2.30.4 SHA3-384-HMAC**

4225 The SHA3-384-HMAC mechanism, denoted **CKM\_SHA3\_384\_HMAC**, is a special case of the general-length SHA3-384-HMAC mechanism.

4227 It has no parameter, and always produces an output of length 48.

## 4228 **2.30.5 SHA3-384 key derivation**

4229 SHA3-384 key derivation, denoted **CKM\_SHA3\_384\_KEY\_DERIVATION**, is the same as the SHA-1 key derivation mechanism in Section 2.20.5, except that it uses the SHA-384 hash function and the relevant length is 48 bytes.

## 4232 **2.30.6 SHA3-384 HMAC key generation**

4233 The SHA3-384-HMAC key generation mechanism, denoted **CKM\_SHA3\_384\_KEY\_GEN**, is a key generation mechanism for NIST's SHA3-384-HMAC.

4235 It does not have a parameter.

4236 The mechanism generates SHA3-384-HMAC keys with a particular length in bytes, as specified in the **CKA\_VALUE\_LEN** attribute of the template for the key.

4238 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new key. Other attributes supported by the SHA3-384-HMAC 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.

4242 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of **CKM\_SHA3\_384\_HMAC** key sizes, in bytes.

## 4244 **2.31 SHA3-512**

4245 *Table 135, SHA-512 Mechanisms vs. Functions*

| <b>Mechanism</b>            | <b>Functions</b>             |                          |                                |               |                           |                          |               |
|-----------------------------|------------------------------|--------------------------|--------------------------------|---------------|---------------------------|--------------------------|---------------|
|                             | <b>Encrypt &amp; Decrypt</b> | <b>Sign &amp; Verify</b> | <b>SR &amp; VR<sup>1</sup></b> | <b>Digest</b> | <b>Gen. Key/ Key Pair</b> | <b>Wrap &amp; Unwrap</b> | <b>Derive</b> |
| CKM_SHA3_512                |                              |                          |                                | ✓             |                           |                          |               |
| CKM_SHA3_512_HMAC_GENERAL   |                              | ✓                        |                                |               |                           |                          |               |
| CKM_SHA3_512_HMAC           |                              | ✓                        |                                |               |                           |                          |               |
| CKM_SHA3_512_KEY_DERIVATION |                              |                          |                                |               |                           |                          | ✓             |
| CKM_SHA3_512_KEY_GEN        |                              |                          |                                | ✓             |                           |                          |               |

4246 **2.31.1 Definitions**

4247 CKM\_SHA3\_512  
4248 CKM\_SHA3\_512\_HMAC  
4249 CKM\_SHA3\_512\_HMAC\_GENERAL  
4250 CKM\_SHA3\_512\_KEY\_DERIVATION  
4251 CKM\_SHA3\_512\_KEY\_GEN  
4252  
4253 CKK\_SHA3\_512\_HMAC

4254 **2.31.2 SHA3-512 digest**

4255 The SHA3-512 mechanism, denoted **CKM\_SHA3\_512**, is a mechanism for message digesting, following  
4256 the Secure Hash 3 Algorithm with a 512-bit message digest defined in FIPS PUB 202.

4257 It does not have a parameter.

4258 Constraints on the length of input and output data are summarized in the following table. For single-part  
4259 digesting, the data and the digest may begin at the same location in memory.

4260 *Table 136, SHA3-512: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 64            |

4261 **2.31.3 General-length SHA3-512-HMAC**

4262 The general-length SHA3-512-HMAC mechanism, denoted **CKM\_SHA3\_512\_HMAC\_GENERAL**, is the  
4263 same as the general-length SHA-1-HMAC mechanism in Section 2.20.4, except that it uses the HMAC  
4264 construction based on the SHA3-512 hash function and length of the output should be in the range 1-  
4265 64. The keys it uses are generic secret keys and CKK\_SHA3\_512\_HMAC. FIPS-198 compliant tokens  
4266 may require the key length to be at least 32 bytes; that is, half the size of the SHA3-512 hash output.

4267

4268 It has a parameter, a CK\_MAC\_GENERAL\_PARAMS, which holds the length in bytes of the desired  
4269 output. This length should be in the range 1-64 (the output size of SHA3-512 is 64 bytes). FIPS-198  
4270 compliant tokens may constrain the output length to be at least 4 or 32 (half the maximum length).  
4271 Signatures (MACs) produced by this mechanism shall be taken from the start of the full 64-byte HMAC  
4272 output.

4273 *Table 137, General-length SHA3-512-HMAC: Key And Data Length*

| Function | Key type                               | Data length | Signature length              |
|----------|----------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret or<br>CKK_SHA3_512_HMAC | Any         | 1-64, depending on parameters |
| C_Verify | generic secret or<br>CKK_SHA3_512_HMAC | Any         | 1-64, depending on parameters |

4274

4275 **2.31.4 SHA3-512-HMAC**

4276 The SHA3-512-HMAC mechanism, denoted **CKM\_SHA3\_512\_HMAC**, is a special case of the general-  
4277 length SHA3-512-HMAC mechanism.

4278 It has no parameter, and always produces an output of length 64.

4279 **2.31.5 SHA3-512 key derivation**

4280 SHA3-512 key derivation, denoted **CKM\_SHA3\_512\_KEY\_DERIVATION**, is the same as the SHA-1 key  
4281 derivation mechanism in Section 2.20.5, except that it uses the SHA-512 hash function and the relevant  
4282 length is 64 bytes.

4283 **2.31.6 SHA3-512 HMAC key generation**

4284 The SHA3-512-HMAC key generation mechanism, denoted **CKM\_SHA3\_512\_KEY\_GEN**, is a key  
4285 generation mechanism for NIST's SHA3-512-HMAC.

4286 It does not have a parameter.

4287 The mechanism generates SHA3-512-HMAC keys with a particular length in bytes, as specified in the  
4288 **CKA\_VALUE\_LEN** attribute of the template for the key.

4289 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
4290 key. Other attributes supported by the SHA3-512-HMAC key type (specifically, the flags indicating which  
4291 functions the key supports) may be specified in the template for the key, or else are assigned default  
4292 initial values.

4293 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
4294 specify the supported range of **CKM\_SHA3\_512\_HMAC** key sizes, in bytes.

4295 **2.32 SHAKE**

4296 *Table 138, SHA-512 Mechanisms vs. Functions*

| Mechanism                    | Functions         |               |                      |        |                    |               |        |
|------------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                              | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SHAKE_128_KEY_DERIVATION |                   |               |                      |        |                    |               | ✓      |
| CKM_SHAKE_256_KEY_DERIVATION |                   |               |                      |        |                    |               | ✓      |

4297 **2.32.1 Definitions**

4298       **CKM\_SHAKE\_128\_KEY\_DERIVATION**

4299       **CKM\_SHAKE\_256\_KEY\_DERIVATION**

4300 **2.32.2 SHAKE Key Derivation**

4301 SHAKE-128 and SHAKE-256 key derivation, denoted **CKM\_SHAKE\_128\_KEY\_DERIVATION** and  
4302 **CKM\_SHAKE\_256\_KEY\_DERIVATION**, implements the SHAKE expansion function defined in FIPS 202  
4303 on the input key.

- 4304     • If no length or key type is provided in the template a **CKR\_TEMPLATE\_INCOMPLETE** error is  
4305       generated.
  - 4306     • If no key type is provided in the template, but a length is, then the key produced by this mechanism  
4307       shall be a generic secret key of the specified length.
  - 4308     • If no length was provided in the template, but a key type is, then that key type must have a well-  
4309       defined length. If it does, then the key produced by this mechanism shall be of the type specified in  
4310       the template. If it doesn't, an error shall be returned.
  - 4311     • If both a key type and a length are provided in the template, the length must be compatible with that  
4312       key type. The key produced by this mechanism shall be of the specified type and length.
- 4313 If a DES, DES2, or CDMF key is derived with this mechanism, the parity bits of the key shall be set  
4314 properly.

- 4315 This mechanism has the following rules about key sensitivity and extractability:
- 4316 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
4317 be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
4318 default value.
- 4319 • If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
4320 shall as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the  
4321 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
4322 **CKA\_SENSITIVE** attribute.
- 4323 • Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then  
4324 the derived key shall, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
4325 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
4326 value from its **CKA\_EXTRACTABLE** attribute.

## 4327 **2.33 Blake2b-160**

4328 *Table 139, Blake2b-160 Mechanisms vs. Functions*

| Mechanism                    | Functions         |               |                      |        |                   |               |        |
|------------------------------|-------------------|---------------|----------------------|--------|-------------------|---------------|--------|
|                              | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/Key Pair | Wrap & Unwrap | Derive |
| CKM_BLAKE2B_160              |                   |               |                      | ✓      |                   |               |        |
| CKM_BLAKE2B_160_HMAC         |                   | ✓             |                      |        |                   |               |        |
| CKM_BLAKE2B_160_HMAC_GENERAL |                   | ✓             |                      |        |                   |               |        |
| CKM_BLAKE2B_160_KEY_DERIVE   |                   |               |                      |        |                   |               | ✓      |
| CKM_BLAKE2B_160_KEY_GEN      |                   |               |                      |        | ✓                 |               |        |

### 4329 **2.33.1 Definitions**

4330 Mechanisms:

- 4331 CKM\_BLAKE2B\_160
- 4332 CKM\_BLAKE2B\_160\_HMAC
- 4333 CKM\_BLAKE2B\_160\_HMAC\_GENERAL
- 4334 CKM\_BLAKE2B\_160\_KEY\_DERIVE
- 4335 CKM\_BLAKE2B\_160\_KEY\_GEN
- 4336 CKK\_BLAKE2B\_160\_HMAC

### 4337 **2.33.2 BLAKE2B-160 digest**

4338 The BLAKE2B-160 mechanism, denoted **CKM\_BLAKE2B\_160**, is a mechanism for message digesting,  
4339 following the Blake2b Algorithm with a 160-bit message digest without a key as defined in [RFC 7693](#).

4340 It does not have a parameter.

4341 Constraints on the length of input and output data are summarized in the following table. For single-part  
4342 digesting, the data and the digest may begin at the same location in memory.

4343 *Table 140, BLAKE2B-160: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 20            |

### 4344 2.33.3 General-length BLAKE2B-160-HMAC

4345 The general-length BLAKE2B-160-HMAC mechanism, denoted  
4346 **CKM\_BLAKE2B\_160\_HMAC\_GENERAL**, is the keyed variant of BLAKE2B-160 and length of the output  
4347 should be in the range 1-20. The keys it uses are generic secret keys and CKK\_BLAKE2B\_160\_HMAC.  
4348 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which holds the length in bytes of the desired  
4349 output. This length should be in the range 1-20 (the output size of BLAKE2B-160 is 20 bytes). Signatures  
4350 (MACs) produced by this mechanism shall be taken from the start of the full 20-byte HMAC output.

4351 *Table 141, General-length BLAKE2B-160-HMAC: Key And Data Length*

| Function | Key type                                      | Data length | Signature length              |
|----------|-----------------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret or<br>CKK_BLAKE2B_160_H<br>MAC | Any         | 1-20, depending on parameters |
| C_Verify | generic secret or<br>CKK_BLAKE2B_160_H<br>MAC | Any         | 1-20, depending on parameters |

### 4352 2.33.4 BLAKE2B-160-HMAC

4353 The BLAKE2B-160-HMAC mechanism, denoted **CKM\_BLAKE2B\_160\_HMAC**, is a special case of the  
4354 general-length BLAKE2B-160-HMAC mechanism.  
4355 It has no parameter, and always produces an output of length 20.

### 4356 2.33.5 BLAKE2B-160 key derivation

4357 BLAKE2B-160 key derivation, denoted **CKM\_BLAKE2B\_160\_KEY\_DERIVE**, is the same as the SHA-1  
4358 key derivation mechanism in Section 2.20.5 except that it uses the BLAKE2B-160 hash function and the  
4359 relevant length is 20 bytes.

### 4360 2.33.6 BLAKE2B-160 HMAC key generation

4361 The BLAKE2B-160-HMAC key generation mechanism, denoted **CKM\_BLAKE2B\_160\_KEY\_GEN**, is a  
4362 key generation mechanism for BLAKE2B-160-HMAC.

4363 It does not have a parameter.

4364 The mechanism generates BLAKE2B-160-HMAC keys with a particular length in bytes, as specified in the  
4365 **CKA\_VALUE\_LEN** attribute of the template for the key.

4366 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
4367 key. Other attributes supported by the BLAKE2B-160-HMAC key type (specifically, the flags indicating  
4368 which functions the key supports) may be specified in the template for the key, or else are assigned  
4369 default initial values.

4370 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
4371 specify the supported range of **CKM\_BLAKE2B\_160\_HMAC** key sizes, in bytes.

## 4372 2.34 BLAKE2B-256

4373 *Table 142, BLAKE2B-256 Mechanisms vs. Functions*

| Mechanism                    | Functions         |               |                      |        |                    |               |        |
|------------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                              | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_BLAKE2B_256              |                   |               |                      | ✓      |                    |               |        |
| CKM_BLAKE2B_256_HMAC_GENERAL |                   | ✓             |                      |        |                    |               |        |
| CKM_BLAKE2B_256_HMAC         |                   | ✓             |                      |        |                    |               |        |
| CKM_BLAKE2B_256_KEY_DERIVE   |                   |               |                      |        |                    |               | ✓      |
| CKM_BLAKE2B_256_KEY_GEN      |                   |               |                      |        | ✓                  |               |        |

### 4374 2.34.1 Definitions

4375 Mechanisms:

4376 CKM\_BLAKE2B\_256

4377 CKM\_BLAKE2B\_256\_HMAC

4378 CKM\_BLAKE2B\_256\_HMAC\_GENERAL

4379 CKM\_BLAKE2B\_256\_KEY\_DERIVE

4380 CKM\_BLAKE2B\_256\_KEY\_GEN

4381 CKK\_BLAKE2B\_256\_HMAC

### 4382 2.34.2 BLAKE2B-256 digest

4383 The BLAKE2B-256 mechanism, denoted **CKM\_BLAKE2B\_256**, is a mechanism for message digesting,  
4384 following the Blake2b Algorithm with a 256-bit message digest without a key as defined in RFC 7693.

4385 It does not have a parameter.

4386 Constraints on the length of input and output data are summarized in the following table. For single-part  
4387 digesting, the data and the digest may begin at the same location in memory.

4388 *Table 143, BLAKE2B-256: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 32            |

### 4389 2.34.3 General-length BLAKE2B-256-HMAC

4390 The general-length BLAKE2B-256-HMAC mechanism, denoted  
4391 **CKM\_BLAKE2B\_256\_HMAC\_GENERAL**, is the keyed variant of Blake2b-256 and length of the output  
4392 should be in the range 1-32. The keys it uses are generic secret keys and CKK\_BLAKE2B\_256\_HMAC.

4393 It has a parameter, a CK\_MAC\_GENERAL\_PARAMS, which holds the length in bytes of the desired  
4394 output. This length should be in the range 1-32 (the output size of BLAKE2B-256 is 32 bytes). Signatures  
4395 (MACs) produced by this mechanism shall be taken from the start of the full 32-byte HMAC output.

4396 Table 144, General-length BLAKE2B-256-HMAC: Key And Data Length

| Function | Key type                                | Data length | Signature length              |
|----------|-----------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret or CKK_BLAKE2B_256_HM_AC | Any         | 1-32, depending on parameters |
| C_Verify | generic secret or CKK_BLAKE2B_256_HM_AC | Any         | 1-32, depending on parameters |

4397 

## 2.34.4 BLAKE2B-256-HMAC

4398 The BLAKE2B-256-HMAC mechanism, denoted **CKM\_BLAKE2B\_256\_HMAC**, is a special case of the  
 4399 general-length BLAKE2B-256-HMAC mechanism in Section 2.22.3.

4400 It has no parameter, and always produces an output of length 32.

4401 

## 2.34.5 BLAKE2B-256 key derivation

4402 BLAKE2B-256 key derivation, denoted **CKM\_BLAKE2B\_256\_KEY\_DERIVE**, is the same as the SHA-1  
 4403 key derivation mechanism in Section 2.20.5, except that it uses the BLAKE2B-256 hash function and the  
 4404 relevant length is 32 bytes.

4405 

## 2.34.6 BLAKE2B-256 HMAC key generation

4406 The BLAKE2B-256-HMAC key generation mechanism, denoted **CKM\_BLAKE2B\_256\_KEY\_GEN**, is a  
 4407 key generation mechanism for BLAKE2B-256-HMAC.

4408 It does not have a parameter.

4409 The mechanism generates BLAKE2B-256-HMAC keys with a particular length in bytes, as specified in the  
 4410 **CKA\_VALUE\_LEN** attribute of the template for the key.

4411 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
 4412 key. Other attributes supported by the BLAKE2B-256-HMAC key type (specifically, the flags indicating  
 4413 which functions the key supports) may be specified in the template for the key, or else are assigned  
 4414 default initial values.

4415 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 4416 specify the supported range of **CKM\_BLAKE2B\_256\_HMAC** key sizes, in bytes.

4417 

## 2.35 BLAKE2B-384

4418 Table 145, BLAKE2B-384 Mechanisms vs. Functions

| Mechanism                    | Functions         |               |                      |        |                    |               |        |
|------------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                              | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_BLAKE2B_384              |                   |               |                      | ✓      |                    |               |        |
| CKM_BLAKE2B_384_HMAC_GENERAL |                   | ✓             |                      |        |                    |               |        |
| CKM_BLAKE2B_384_HMAC         |                   | ✓             |                      |        |                    |               |        |
| CKM_BLAKE2B_384_KEY_DERIVE   |                   |               |                      |        |                    |               | ✓      |
| CKM_BLAKE2B_384_KEY_GEN      |                   |               |                      | ✓      |                    |               |        |

4419 **2.35.1 Definitions**

4420 CKM\_BLAKE2B\_384  
4421 CKM\_BLAKE2B\_384\_HMAC  
4422 CKM\_BLAKE2B\_384\_HMAC\_GENERAL  
4423 CKM\_BLAKE2B\_384\_KEY\_DERIVE  
4424 CKM\_BLAKE2B\_384\_KEY\_GEN  
4425 CKK\_BLAKE2B\_384\_HMAC

4426 **2.35.2 BLAKE2B-384 digest**

4427 The BLAKE2B-384 mechanism, denoted **CKM\_BLAKE2B\_384**, is a mechanism for message digesting,  
4428 following the Blake2b Algorithm with a 384-bit message digest without a key as defined in RFC 7693.

4429 It does not have a parameter.

4430 Constraints on the length of input and output data are summarized in the following table. For single-part  
4431 digesting, the data and the digest may begin at the same location in memory.

4432 *Table 146, BLAKE2B-384: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 48            |

4433 **2.35.3 General-length BLAKE2B-384-HMAC**

4434 The general-length BLAKE2B-384-HMAC mechanism, denoted  
4435 **CKM\_BLAKE2B\_384\_HMAC\_GENERAL**, is the keyed variant of the Blake2b-384 hash function and  
4436 length of the output should be in the range 1-48. The keys it uses are generic secret keys and  
4437 CKK\_BLAKE2B\_384\_HMAC.

4438

4439 It has a parameter, a CK\_MAC\_GENERAL\_PARAMS, which holds the length in bytes of the desired  
4440 output. This length should be in the range 1-48 (the output size of BLAKE2B-384 is 48 bytes). Signatures  
4441 (MACs) produced by this mechanism shall be taken from the start of the full 48-byte HMAC output.

4442 *Table 147, General-length BLAKE2B-384-HMAC: Key And Data Length*

| Function | Key type                                      | Data length | Signature length              |
|----------|-----------------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret or<br>CKK_BLAKE2B_384_H<br>MAC | Any         | 1-48, depending on parameters |
| C_Verify | generic secret or<br>CKK_BLAKE2B_384_H<br>MAC | Any         | 1-48, depending on parameters |

4443

4444 **2.35.4 BLAKE2B-384-HMAC**

4445 The BLAKE2B-384-HMAC mechanism, denoted **CKM\_BLAKE2B\_384\_HMAC**, is a special case of the  
4446 general-length BLAKE2B-384-HMAC mechanism.

4447 It has no parameter, and always produces an output of length 48.

## 4448 2.35.5 BLAKE2B-384 key derivation

4449 BLAKE2B-384 key derivation, denoted **CKM\_BLAKE2B\_384\_KEY\_DERIVE**, is the same as the SHA-1  
4450 key derivation mechanism in Section 2.20.5, except that it uses the SHA-384 hash function and the  
4451 relevant length is 48 bytes.

## 4452 2.35.6 BLAKE2B-384 HMAC key generation

4453 The BLAKE2B-384-HMAC key generation mechanism, denoted **CKM\_BLAKE2B\_384\_KEY\_GEN**, is a  
4454 key generation mechanism for NIST's BLAKE2B-384-HMAC.

4455 It does not have a parameter.

4456 The mechanism generates BLAKE2B-384-HMAC keys with a particular length in bytes, as specified in the  
4457 **CKA\_VALUE\_LEN** attribute of the template for the key.

4458 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
4459 key. Other attributes supported by the BLAKE2B-384-HMAC key type (specifically, the flags indicating  
4460 which functions the key supports) may be specified in the template for the key, or else are assigned  
4461 default initial values.

4462 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
4463 specify the supported range of **CKM\_BLAKE2B\_384\_HMAC** key sizes, in bytes.

## 4464 2.36 BLAKE2B-512

4465 *Table 148, SHA-512 Mechanisms vs. Functions*

| Mechanism                    | Functions         |               |                      |        |                    |               |        |
|------------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                              | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_BLAKE2B_512              |                   |               |                      | ✓      |                    |               |        |
| CKM_BLAKE2B_512_HMAC_GENERAL |                   | ✓             |                      |        |                    |               |        |
| CKM_BLAKE2B_512_HMAC         |                   | ✓             |                      |        |                    |               |        |
| CKM_BLAKE2B_512_KEY_DERIVE   |                   |               |                      |        |                    |               | ✓      |
| CKM_BLAKE2B_512_KEY_GEN      |                   |               |                      | ✓      |                    |               |        |

## 4466 2.36.1 Definitions

4467 CKM\_BLAKE2B\_512

4468 CKM\_BLAKE2B\_512\_HMAC

4469 CKM\_BLAKE2B\_512\_HMAC\_GENERAL

4470 CKM\_BLAKE2B\_512\_KEY\_DERIVE

4471 CKM\_BLAKE2B\_512\_KEY\_GEN

4472 CKK\_BLAKE2B\_512\_HMAC

## 4473 2.36.2 BLAKE2B-512 digest

4474 The BLAKE2B-512 mechanism, denoted **CKM\_BLAKE2B\_512**, is a mechanism for message digesting,  
4475 following the Blake2b Algorithm with a 512-bit message digest defined in RFC 7693.

4476 It does not have a parameter.

4477 Constraints on the length of input and output data are summarized in the following table. For single-part  
4478 digesting, the data and the digest may begin at the same location in memory.

4479 *Table 149, BLAKE2B-512: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | any          | 64            |

### 4480 2.36.3 General-length BLAKE2B-512-HMAC

4481 The general-length BLAKE2B-512-HMAC mechanism, denoted  
4482 **CKM\_BLAKE2B\_512\_HMAC\_GENERAL**, is the keyed variant of the BLAKE2B-512 hash function and  
4483 length of the output should be in the range 1-64. The keys it uses are generic secret keys and  
4484 CKK\_BLAKE2B\_512\_HMAC.

4485

4486 It has a parameter, a CK\_MAC\_GENERAL\_PARAMS, which holds the length in bytes of the desired  
4487 output. This length should be in the range 1-64 (the output size of BLAKE2B-512 is 64 bytes). Signatures  
4488 (MACs) produced by this mechanism shall be taken from the start of the full 64-byte HMAC output.

4489 *Table 150, General-length BLAKE2B-512-HMAC: Key And Data Length*

| Function | Key type                                      | Data length | Signature length              |
|----------|-----------------------------------------------|-------------|-------------------------------|
| C_Sign   | generic secret or<br>CKK_BLAKE2B_512_HM<br>AC | Any         | 1-64, depending on parameters |
| C_Verify | generic secret or<br>CKK_BLAKE2B_512_HM<br>AC | Any         | 1-64, depending on parameters |

4490

### 4491 2.36.4 BLAKE2B-512-HMAC

4492 The BLAKE2B-512-HMAC mechanism, denoted **CKM\_BLAKE2B\_512\_HMAC**, is a special case of the  
4493 general-length BLAKE2B-512-HMAC mechanism.

4494 It has no parameter, and always produces an output of length 64.

### 4495 2.36.5 BLAKE2B-512 key derivation

4496 BLAKE2B-512 key derivation, denoted **CKM\_BLAKE2B\_512\_KEY\_DERIVE**, is the same as the SHA-1  
4497 key derivation mechanism in Section 2.20.5, except that it uses the Blake2b-512 hash function and the  
4498 relevant length is 64 bytes.

### 4499 2.36.6 BLAKE2B-512 HMAC key generation

4500 The BLAKE2B-512-HMAC key generation mechanism, denoted **CKM\_BLAKE2B\_512\_KEY\_GEN**, is a  
4501 key generation mechanism for NIST's BLAKE2B-512-HMAC.

4502 It does not have a parameter.

4503 The mechanism generates BLAKE2B-512-HMAC keys with a particular length in bytes, as specified in the  
4504 **CKA\_VALUE\_LEN** attribute of the template for the key.

4505 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
4506 key. Other attributes supported by the BLAKE2B-512-HMAC key type (specifically, the flags indicating  
4507 which functions the key supports) may be specified in the template for the key, or else are assigned  
4508 default initial values.

4509 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
4510 specify the supported range of **CKM\_BLAKE2B\_512\_HMAC** key sizes, in bytes.

4511

## 4512 2.37 PKCS #5 and PKCS #5-style password-based encryption (PBE)

4513 The mechanisms in this section are for generating keys and IVs for performing password-based  
 4514 encryption. The method used to generate keys and IVs is specified in PKCS #5.

4515 *Table 151, PKCS 5 Mechanisms vs. Functions*

| Mechanism                   | Functions         |               |                      |        |                    |               |        |
|-----------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                             | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_PBE_SHA1_DES3_EDE_CBC   |                   |               |                      |        | ✓                  |               |        |
| CKM_PBE_SHA1_DES2_EDE_CBC   |                   |               |                      |        | ✓                  |               |        |
| CKM_PBA_SHA1_WITH_SHA1_HMAC |                   |               |                      |        | ✓                  |               |        |
| CKM_PKCS5_PBKD2             |                   |               |                      |        | ✓                  |               |        |

### 4516 2.37.1 Definitions

4517 Mechanisms:

4518 CKM\_PBE\_SHA1\_DES3\_EDE\_CBC  
 4519 CKM\_PBE\_SHA1\_DES2\_EDE\_CBC  
 4520 CKM\_PKCS5\_PBKD2  
 4521 CKM\_PBA\_SHA1\_WITH\_SHA1\_HMAC

### 4522 2.37.2 Password-based encryption/authentication mechanism parameters

#### 4523 ♦ CK\_PBE\_PARAMS; CK\_PBE\_PARAMS\_PTR

4524 **CK\_PBE\_PARAMS** is a structure which provides all of the necessary information required by the  
 4525 CKM\_PBE mechanisms (see PKCS #5 and PKCS #12 for information on the PBE generation  
 4526 mechanisms) and the CKM\_PBA\_SHA1\_WITH\_SHA1\_HMAC mechanism. It is defined as follows:

```
4527 typedef struct CK_PBE_PARAMS {
4528 CK_BYTE_PTR pInitVector;
4529 CK_UTF8CHAR_PTR pPassword;
4530 CK ULONG ulPasswordLen;
4531 CK_BYTE_PTR pSalt;
4532 CK ULONG ulSaltLen;
4533 CK ULONG ulIteration;
4534 } CK_PBE_PARAMS;
```

4535  
 4536 The fields of the structure have the following meanings:

4537 *pInitVector* pointer to the location that receives the 8-byte initialization vector  
 4538 (IV), if an IV is required;

4539 *pPassword* points to the password to be used in the PBE key generation;

4540 *ulPasswordLen* length in bytes of the password information;

4541                    *pSalt*        *points to the salt to be used in the PBE key generation;*  
 4542                    *ulSaltLen*      *length in bytes of the salt information;*  
 4543                    *ullIteration*    *number of iterations required for the generation.*

4544 **CK\_PBE\_PARAMS\_PTR** is a pointer to a **CK\_PBE\_PARAMS**.

### 4545 2.37.3 PKCS #5 PBKDF2 key generation mechanism parameters

4546 ◆ **CK\_PKCS5\_PBKD2\_PSEUDO\_RANDOM\_FUNCTION\_TYPE;**  
 4547 **CK\_PKCS5\_PBKD2\_PSEUDO\_RANDOM\_FUNCTION\_TYPE\_PTR**

4548 **CK\_PKCS5\_PBKD2\_PSEUDO\_RANDOM\_FUNCTION\_TYPE** is used to indicate the Pseudo-Random  
 4549 Function (PRF) used to generate key bits using PKCS #5 PBKDF2. It is defined as follows:

4550        `typedef CK ULONG CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE;`

4551

4552 The following PRFs are defined in PKCS #5 v2.1. The following table lists the defined functions.

4553 *Table 152, PKCS #5 PBKDF2 Key Generation: Pseudo-random functions*

| PRF Identifier                  | Value        | Parameter Type                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKP_PKCS5_PBKD2_HMAC_SHA1       | 0x00000001UL | No Parameter. <i>pPrfData</i> must be NULL and <i>ulPrfDataLen</i> must be zero.                                                                                                                                                                                                                                                                                                            |
| CKP_PKCS5_PBKD2_HMAC_GOSTR3411  | 0x00000002UL | This PRF uses GOST R34.11-94 hash to produce secret key value. <i>pPrfData</i> should point to DER-encoded OID, indicating GOSTR34.11-94 parameters. <i>ulPrfDataLen</i> holds encoded OID length in bytes. If <i>pPrfData</i> is set to <code>NULL_PTR</code> , then <i>id-GostR3411-94-CryptoProParamSet</i> parameters will be used (RFC 4357, 11.2), and <i>ulPrfDataLen</i> must be 0. |
| CKP_PKCS5_PBKD2_HMAC_SHA224     | 0x00000003UL | No Parameter. <i>pPrfData</i> must be NULL and <i>ulPrfDataLen</i> must be zero.                                                                                                                                                                                                                                                                                                            |
| CKP_PKCS5_PBKD2_HMAC_SHA256     | 0x00000004UL | No Parameter. <i>pPrfData</i> must be NULL and <i>ulPrfDataLen</i> must be zero.                                                                                                                                                                                                                                                                                                            |
| CKP_PKCS5_PBKD2_HMAC_SHA384     | 0x00000005UL | No Parameter. <i>pPrfData</i> must be NULL and <i>ulPrfDataLen</i> must be zero.                                                                                                                                                                                                                                                                                                            |
| CKP_PKCS5_PBKD2_HMAC_SHA512     | 0x00000006UL | No Parameter. <i>pPrfData</i> must be NULL and <i>ulPrfDataLen</i> must be zero.                                                                                                                                                                                                                                                                                                            |
| CKP_PKCS5_PBKD2_HMAC_SHA512_224 | 0x00000007UL | No Parameter. <i>pPrfData</i> must be NULL and <i>ulPrfDataLen</i> must be zero.                                                                                                                                                                                                                                                                                                            |

|                                 |              |                                                                                  |
|---------------------------------|--------------|----------------------------------------------------------------------------------|
| CKP_PKCS5_PBKD2_HMAC_SHA512_256 | 0x00000008UL | No Parameter. <i>pPrfData</i> must be NULL and <i>ulPrfDataLen</i> must be zero. |
|---------------------------------|--------------|----------------------------------------------------------------------------------|

4554 CK\_PKCS5\_PBKD2\_PSEUDO\_RANDOM\_FUNCTION\_TYPE\_PTR is a pointer to a  
 4555 CK\_PKCS5\_PBKD2\_PSEUDO\_RANDOM\_FUNCTION\_TYPE.

4556

- 4557 ◆ CK\_PKCS5\_PBKDF2\_SALT\_SOURCE\_TYPE;  
 4558 CK\_PKCS5\_PBKDF2\_SALT\_SOURCE\_TYPE\_PTR

4559 CK\_PKCS5\_PBKDF2\_SALT\_SOURCE\_TYPE is used to indicate the source of the salt value when  
 4560 deriving a key using PKCS #5 PBKDF2. It is defined as follows:

4561   typedef CK ULONG CK\_PKCS5\_PBKDF2\_SALT\_SOURCE\_TYPE;

4562

4563 The following salt value sources are defined in PKCS #5 v2.1. The following table lists the defined  
 4564 sources along with the corresponding data type for the *pSaltSourceData* field in the  
 4565 CK\_PKCS5\_PBKD2\_PARAMS2 structure defined below.

4566 *Table 153, PKCS #5 PBKDF2 Key Generation: Salt sources*

| Source Identifier  | Value      | Data Type                                                |
|--------------------|------------|----------------------------------------------------------|
| CKZ_SALT_SPECIFIED | 0x00000001 | Array of CK_BYTE containing the value of the salt value. |

4567 CK\_PKCS5\_PBKDF2\_SALT\_SOURCE\_TYPE\_PTR is a pointer to a  
 4568 CK\_PKCS5\_PBKDF2\_SALT\_SOURCE\_TYPE.

- 4569 ◆ CK\_PKCS5\_PBKD2\_PARAMS2; CK\_PKCS5\_PBKD2\_PARAMS2\_PTR

4570 CK\_PKCS5\_PBKD2\_PARAMS2 is a structure that provides the parameters to the  
 4571 CKM\_PKCS5\_PBKD2 mechanism. The structure is defined as follows:

```
4572 typedef struct CK_PKCS5_PBKD2_PARAMS2 {
4573 CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE saltSource;
4574 CK_VOID_PTR pSaltSourceData;
4575 CK ULONG ulSaltSourceDataLen;
4576 CK ULONG iterations;
4577 CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE prf;
4578 CK_VOID_PTR pPrfData;
4579 CK ULONG ulPrfDataLen;
4580 CK_UTF8CHAR_PTR pPassword;
4581 CK ULONG ulPasswordLen;
4582 } CK_PKCS5_PBKD2_PARAMS2;
```

4583

4584 The fields of the structure have the following meanings:

4585            *saltSource*      *source of the salt value*

4586            *pSaltSourceData*   *data used as the input for the salt source*

4587            *ulSaltSourceDataLen*   *length of the salt source input*

4588                   *iterations*     *number of iterations to perform when generating each block of*  
4589                   *random data*

4590                   *prf*         *pseudo-random function used to generate the key*

4591                   *pPrfData*    *data used as the input for PRF in addition to the salt value*

4592                   *ulPrfDataLen*   *length of the input data for the PRF*

4593                   *pPassword*    *points to the password to be used in the PBE key generation*

4594                   *ulPasswordLen*   *length in bytes of the password information*

4595 **CK\_PKCS5\_PBKD2\_PARAMS2\_PTR** is a pointer to a **CK\_PKCS5\_PBKD2\_PARAMS2**.

#### 4596 **2.37.4 PKCS #5 PBKD2 key generation**

4597 PKCS #5 PBKDF2 key generation, denoted **CKM\_PKCS5\_PBKD2**, is a mechanism used for generating  
4598 a secret key from a password and a salt value. This functionality is defined in PKCS#5 as PBKDF2.  
4599 It has a parameter, a **CK\_PKCS5\_PBKD2\_PARAMS2** structure. The parameter specifies the salt value  
4600 source, pseudo-random function, and iteration count used to generate the new key.  
4601 Since this mechanism can be used to generate any type of secret key, new key templates must contain  
4602 the **CKA\_KEY\_TYPE** and **CKA\_VALUE\_LEN** attributes. If the key type has a fixed length the  
4603 **CKA\_VALUE\_LEN** attribute may be omitted.

#### 4604 **2.38 PKCS #12 password-based encryption/authentication 4605 mechanisms**

4606 The mechanisms in this section are for generating keys and IVs for performing password-based  
4607 encryption or authentication. The method used to generate keys and IVs is based on a method that was  
4608 specified in PKCS #12.

4609 We specify here a general method for producing various types of pseudo-random bits from a password,  
4610 *p*; a string of salt bits, *s*; and an iteration count, *c*. The “type” of pseudo-random bits to be produced is  
4611 identified by an identification byte, *ID*, the meaning of which will be discussed later.

4612 Let *H* be a hash function built around a compression function  $f: \mathbb{Z}_2^u \times \mathbb{Z}_2^v \rightarrow \mathbb{Z}_2^u$  (that is, *H* has a chaining  
4613 variable and output of length *u* bits, and the message input to the compression function of *H* is *v* bits).  
4614 For MD2 and MD5, *u*=128 and *v*=512; for SHA-1, *u*=160 and *v*=512.

4615 We assume here that *u* and *v* are both multiples of 8, as are the lengths in bits of the password and salt  
4616 strings and the number *n* of pseudo-random bits required. In addition, *u* and *v* are of course nonzero.

- 4617 1. Construct a string, *D* (the “diversifier”), by concatenating *v*/8 copies of *ID*.
- 4618 2. Concatenate copies of the salt together to create a string *S* of length  $v - \lceil s/v \rceil$  bits (the final copy of the  
4619 salt may be truncated to create *S*). Note that if the salt is the empty string, then so is *S*.
- 4620 3. Concatenate copies of the password together to create a string *P* of length  $v - \lceil p/v \rceil$  bits (the final copy  
4621 of the password may be truncated to create *P*). Note that if the password is the empty string, then so  
4622 is *P*.
- 4623 4. Set *I*=*S||P* to be the concatenation of *S* and *P*.
- 4624 5. Set *j*= $\lceil n/u \rceil$ .
- 4625 6. For *i*=1, 2, ..., *j*, do the following:
  - 4626 a. Set  $A_i = H^c(D||I)$ , the *c*<sup>th</sup> hash of *D||I*. That is, compute the hash of *D||I*; compute the hash of  
4627 that hash; etc.; continue in this fashion until a total of *c* hashes have been computed, each on  
4628 the result of the previous hash.

- 4629            b. Concatenate copies of  $A_i$  to create a string  $B$  of length  $v$  bits (the final copy of  $A_i$  may be  
4630            truncated to create  $B$ ).  
4631            c. Treating  $I$  as a concatenation  $I_0, I_1, \dots, I_{k-1}$  of  $v$ -bit blocks, where  $k=\lceil s/v \rceil + \lceil p/v \rceil$ , modify  $I$  by  
4632            setting  $I_j = (I_j + B + 1) \bmod 2^v$  for each  $j$ . To perform this addition, treat each  $v$ -bit block as a  
4633            binary number represented most-significant bit first.  
4634      7. Concatenate  $A_1, A_2, \dots, A_k$  together to form a pseudo-random bit string,  $A$ .  
4635      8. Use the first  $n$  bits of  $A$  as the output of this entire process.  
4636 When the password-based encryption mechanisms presented in this section are used to generate a key  
4637 and IV (if needed) from a password, salt, and an iteration count, the above algorithm is used. To  
4638 generate a key, the identifier byte  $ID$  is set to the value 1; to generate an IV, the identifier byte  $ID$  is set to  
4639 the value 2.  
4640 When the password based authentication mechanism presented in this section is used to generate a key  
4641 from a password, salt, and an iteration count, the above algorithm is used. The identifier byte  $ID$  is set to  
4642 the value 3.

### 4643 2.38.1 SHA-1-PBE for 3-key triple-DES-CBC

4644 SHA-1-PBE for 3-key triple-DES-CBC, denoted **CKM\_PBE\_SHA1\_DES3\_EDE\_CBC**, is a mechanism  
4645 used for generating a 3-key triple-DES secret key and IV from a password and a salt value by using the  
4646 SHA-1 digest algorithm and an iteration count. The method used to generate the key and IV is described  
4647 above. Each byte of the key produced will have its low-order bit adjusted, if necessary, so that a valid 3-  
4648 key triple-DES key with proper parity bits is obtained.

4649 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
4650 key generation process and the location of the application-supplied buffer which will receive the 8-byte IV  
4651 generated by the mechanism.

4652 The key and IV produced by this mechanism will typically be used for performing password-based  
4653 encryption.

### 4654 2.38.2 SHA-1-PBE for 2-key triple-DES-CBC

4655 SHA-1-PBE for 2-key triple-DES-CBC, denoted **CKM\_PBE\_SHA1\_DES2\_EDE\_CBC**, is a mechanism  
4656 used for generating a 2-key triple-DES secret key and IV from a password and a salt value by using the  
4657 SHA-1 digest algorithm and an iteration count. The method used to generate the key and IV is described  
4658 above. Each byte of the key produced will have its low-order bit adjusted, if necessary, so that a valid 2-  
4659 key triple-DES key with proper parity bits is obtained.

4660 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
4661 key generation process and the location of the application-supplied buffer which will receive the 8-byte IV  
4662 generated by the mechanism.

4663 The key and IV produced by this mechanism will typically be used for performing password-based  
4664 encryption.

### 4665 2.38.3 SHA-1-PBA for SHA-1-HMAC

4666 SHA-1-PBA for SHA-1-HMAC, denoted **CKM\_PBA\_SHA1\_WITH\_SHA1\_HMAC**, is a mechanism used  
4667 for generating a 160-bit generic secret key from a password and a salt value by using the SHA-1 digest  
4668 algorithm and an iteration count. The method used to generate the key is described above.

4669 It has a parameter, a **CK\_PBE\_PARAMS** structure. The parameter specifies the input information for the  
4670 key generation process. The parameter also has a field to hold the location of an application-supplied  
4671 buffer which will receive an IV; for this mechanism, the contents of this field are ignored, since  
4672 authentication with SHA-1-HMAC does not require an IV.

4673 The key generated by this mechanism will typically be used for computing a SHA-1 HMAC to perform  
4674 password-based authentication (not *password-based encryption*). At the time of this writing, this is  
4675 primarily done to ensure the integrity of a PKCS #12 PDU.

## 4676 2.39 SSL

4677 Table 154,SSL Mechanisms vs. Functions

| Mechanism                     | Functions         |               |                      |        |                    |               |        |
|-------------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                               | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen - Key/Key Pair | Wrap & Unwrap | Derive |
| CKM_SSL3_PRE_MASTER_KEY_GEN   |                   |               |                      |        | ✓                  |               |        |
| CKM_TLS_PRE_MASTER_KEY_GEN    |                   |               |                      |        | ✓                  |               |        |
| CKM_SSL3_MASTER_KEY_DERIVE    |                   |               |                      |        |                    |               | ✓      |
| CKM_SSL3_MASTER_KEY_DERIVE_DH |                   |               |                      |        |                    |               | ✓      |
| CKM_SSL3_KEY_AND_MAC_DERIVE   |                   |               |                      |        |                    |               | ✓      |
| CKM_SSL3_MD5_MAC              |                   |               | ✓                    |        |                    |               |        |
| CKM_SSL3_SHA1_MAC             |                   | ✓             |                      |        |                    |               |        |

### 4678 2.39.1 Definitions

4679 Mechanisms:

4680 CKM\_SSL3\_PRE\_MASTER\_KEY\_GEN  
4681 CKM\_TLS\_PRE\_MASTER\_KEY\_GEN  
4682 CKM\_SSL3\_MASTER\_KEY\_DERIVE  
4683 CKM\_SSL3\_KEY\_AND\_MAC\_DERIVE  
4684 CKM\_SSL3\_MASTER\_KEY\_DERIVE\_DH  
4685 CKM\_SSL3\_MD5\_MAC  
4686 CKM\_SSL3\_SHA1\_MAC

### 4687 2.39.2 SSL mechanism parameters

#### 4688 ♦ CK\_SSL3\_RANDOM\_DATA

4689 CK\_SSL3\_RANDOM\_DATA is a structure which provides information about the random data of a client  
4690 and a server in an SSL context. This structure is used by both the CKM\_SSL3\_MASTER\_KEY\_DERIVE  
4691 and the CKM\_SSL3\_KEY\_AND\_MAC\_DERIVE mechanisms. It is defined as follows:

```
4692 typedef struct CK_SSL3_RANDOM_DATA {
4693 CK_BYTE_PTR pClientRandom;
4694 CK ULONG ulClientRandomLen;
4695 CK_BYTE_PTR pServerRandom;
4696 CK ULONG ulServerRandomLen;
4697 } CK_SSL3_RANDOM_DATA;
```

4698

4699 The fields of the structure have the following meanings:

4700 pClientRandom pointer to the client's random data

4701 ulClientRandomLen length in bytes of the client's random data

4702                   *pServerRandom*     pointer to the server's random data  
4703                   *ulServerRandomLen*   length in bytes of the server's random data  
  
4704   ◆ **CK\_SSL3\_MASTER\_KEY\_DERIVE\_PARAMS;**  
4705   **CK\_SSL3\_MASTER\_KEY\_DERIVE\_PARAMS\_PTR**

4706 **CK\_SSL3\_MASTER\_KEY\_DERIVE\_PARAMS** is a structure that provides the parameters to the  
4707 **CKM\_SSL3\_MASTER\_KEY\_DERIVE** mechanism. It is defined as follows:

```
4708 typedef struct CK_SSL3_MASTER_KEY_DERIVE_PARAMS {
4709 CK_SSL3_RANDOM_DATA RandomInfo;
4710 CK_VERSION_PTR pVersion;
4711 } CK_SSL3_MASTER_KEY_DERIVE_PARAMS;
```

4712  
4713 The fields of the structure have the following meanings:  
4714                   *RandomInfo*     client's and server's random data information.

4715                   *pVersion*      pointer to a **CK\_VERSION** structure which receives the SSL  
4716                                          protocol version information

4717 **CK\_SSL3\_MASTER\_KEY\_DERIVE\_PARAMS\_PTR** is a pointer to a  
4718 **CK\_SSL3\_MASTER\_KEY\_DERIVE\_PARAMS**.

4719   ◆ **CK\_SSL3\_KEY\_MAT\_OUT; CK\_SSL3\_KEY\_MAT\_OUT\_PTR**

4720 **CK\_SSL3\_KEY\_MAT\_OUT** is a structure that contains the resulting key handles and initialization vectors  
4721 after performing a C\_DeriveKey function with the **CKM\_SSL3\_KEY\_AND\_MAC\_DERIVE** mechanism. It  
4722 is defined as follows:

```
4723 typedef struct CK_SSL3_KEY_MAT_OUT {
4724 CK_OBJECT_HANDLE hClientMacSecret;
4725 CK_OBJECT_HANDLE hServerMacSecret;
4726 CK_OBJECT_HANDLE hClientKey;
4727 CK_OBJECT_HANDLE hServerKey;
4728 CK_BYTE_PTR pIVClient;
4729 CK_BYTE_PTR pIVServer;
4730 } CK_SSL3_KEY_MAT_OUT;
```

4731  
4732 The fields of the structure have the following meanings:  
4733                   *hClientMacSecret*   key handle for the resulting Client MAC Secret key

4734                   *hServerMacSecret*   key handle for the resulting Server MAC Secret key

4735                   *hClientKey*       key handle for the resulting Client Secret key

4736                   *hServerKey*      key handle for the resulting Server Secret key

4737                   *pIVClient*      pointer to a location which receives the initialization vector (IV)  
4738                                          created for the client (if any)

4739                   *pIVServer*     pointer to a location which receives the initialization vector (IV)  
4740                                          created for the server (if any)

4741 **CK\_SSL3\_KEY\_MAT\_OUT\_PTR** is a pointer to a **CK\_SSL3\_KEY\_MAT\_OUT**.

4742 ◆ **CK\_SSL3\_KEY\_MAT\_PARAMS; CK\_SSL3\_KEY\_MAT\_PARAMS\_PTR**

4743 **CK\_SSL3\_KEY\_MAT\_PARAMS** is a structure that provides the parameters to the  
 4744 **CKM\_SSL3\_KEY\_AND\_MAC\_DERIVE** mechanism. It is defined as follows:

```

4745 typedef struct CK_SSL3_KEY_MAT_PARAMS {
4746 CK ULONG ulMacSizeInBits;
4747 CK ULONG ulKeySizeInBits;
4748 CK ULONG ullIVSizeInBits;
4749 CK_BBOOL bIsExport;
4750 CK_SSL3_RANDOM_DATA RandomInfo;
4751 CK_SSL3_KEY_MAT_OUT_PTR pReturnedKeyMaterial;
4752 } CK_SSL3_KEY_MAT_PARAMS;

```

4753

4754 The fields of the structure have the following meanings:

|                                                                                  |                                                                                                                                            |
|----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 4755 <ul style="list-style-type: none"><li><b>ulMacSizeInBits</b></li></ul>      | <i>the length (in bits) of the MACing keys agreed upon during the protocol handshake phase</i>                                             |
| 4757 <ul style="list-style-type: none"><li><b>ulKeySizeInBits</b></li></ul>      | <i>the length (in bits) of the secret keys agreed upon during the protocol handshake phase</i>                                             |
| 4759 <ul style="list-style-type: none"><li><b>ullIVSizeInBits</b></li></ul>      | <i>the length (in bits) of the IV agreed upon during the protocol handshake phase. If no IV is required, the length should be set to 0</i> |
| 4761 <ul style="list-style-type: none"><li><b>bIsExport</b></li></ul>            | <i>a Boolean value which indicates whether the keys have to be derived for an export version of the protocol</i>                           |
| 4763 <ul style="list-style-type: none"><li><b>RandomInfo</b></li></ul>           | <i>client's and server's random data information.</i>                                                                                      |
| 4764 <ul style="list-style-type: none"><li><b>pReturnedKeyMaterial</b></li></ul> | <i>points to a CK_SSL3_KEY_MAT_OUT structures which receives the handles for the keys generated and the IVs</i>                            |

4766 **CK\_SSL3\_KEY\_MAT\_PARAMS\_PTR** is a pointer to a **CK\_SSL3\_KEY\_MAT\_PARAMS**.

4767 **2.39.3 Pre-master key generation**

4768 Pre-master key generation in SSL 3.0, denoted **CKM\_SSL3\_PRE\_MASTER\_KEY\_GEN**, is a mechanism  
 4769 which generates a 48-byte generic secret key. It is used to produce the "pre\_master" key used in SSL  
 4770 version 3.0 for RSA-like cipher suites.

4771 It has one parameter, a **CK\_VERSION** structure, which provides the client's SSL version number.

4772 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
 4773 key (as well as the **CKA\_VALUE\_LEN** attribute, if it is not supplied in the template). Other attributes may  
 4774 be specified in the template, or else are assigned default values.

4775 The template sent along with this mechanism during a **C\_GenerateKey** call may indicate that the object  
 4776 class is **CKO\_SECRET\_KEY**, the key type is **CKK\_GENERIC\_SECRET**, and the **CKA\_VALUE\_LEN**  
 4777 attribute has value 48. However, since these facts are all implicit in the mechanism, there is no need to  
 4778 specify any of them.

4779 For this mechanism, the **ulMinKeySize** and **ulMaxKeySize** fields of the **CK\_MECHANISM\_INFO** structure  
 4780 both indicate 48 bytes.

4781 **CKM\_TLS\_PRE\_MASTER\_KEY\_GEN** has identical functionality as  
4782 **CKM\_SSL3\_PRE\_MASTER\_KEY\_GEN**. It exists only for historical reasons, please use  
4783 **CKM\_SSL3\_PRE\_MASTER\_KEY\_GEN** instead.

#### 4784 **2.39.4 Master key derivation**

4785 Master key derivation in SSL 3.0, denoted **CKM\_SSL3\_MASTER\_KEY\_DERIVE**, is a mechanism used  
4786 to derive one 48-byte generic secret key from another 48-byte generic secret key. It is used to produce  
4787 the "master\_secret" key used in the SSL protocol from the "pre\_master" key. This mechanism returns the  
4788 value of the client version, which is built into the "pre\_master" key as well as a handle to the derived  
4789 "master\_secret" key.

4790 It has a parameter, a **CK\_SSL3\_MASTER\_KEY\_DERIVE\_PARAMS** structure, which allows for the  
4791 passing of random data to the token as well as the returning of the protocol version number which is part  
4792 of the pre-master key. This structure is defined in Section 2.39.

4793 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
4794 key (as well as the **CKA\_VALUE\_LEN** attribute, if it is not supplied in the template). Other attributes may  
4795 be specified in the template; otherwise they are assigned default values.

4796 The template sent along with this mechanism during a **C\_DeriveKey** call may indicate that the object  
4797 class is **CKO\_SECRET\_KEY**, the key type is **CKK\_GENERIC\_SECRET**, and the **CKA\_VALUE\_LEN**  
4798 attribute has value 48. However, since these facts are all implicit in the mechanism, there is no need to  
4799 specify any of them.

4800 This mechanism has the following rules about key sensitivity and extractability:

- 4801 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
4802 be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
4803 default value.
- 4804 • If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
4805 will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the  
4806 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
4807 **CKA\_SENSITIVE** attribute.
- 4808 • Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
4809 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
4810 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
4811 value from its **CKA\_EXTRACTABLE** attribute.

4812 For this mechanism, the ulMinKeySize and ulMaxKeySize fields of the **CK\_MECHANISM\_INFO** structure  
4813 both indicate 48 bytes.

4814 Note that the **CK\_VERSION** structure pointed to by the **CK\_SSL3\_MASTER\_KEY\_DERIVE\_PARAMS**  
4815 structure's *pVersion* field will be modified by the **C\_DeriveKey** call. In particular, when the call returns,  
4816 this structure will hold the SSL version associated with the supplied pre\_master key.

4817 Note that this mechanism is only useable for cipher suites that use a 48-byte "pre\_master" secret with an  
4818 embedded version number. This includes the RSA cipher suites, but excludes the Diffie-Hellman cipher  
4819 suites.

#### 4820 **2.39.5 Master key derivation for Diffie-Hellman**

4821 Master key derivation for Diffie-Hellman in SSL 3.0, denoted **CKM\_SSL3\_MASTER\_KEY\_DERIVE\_DH**,  
4822 is a mechanism used to derive one 48-byte generic secret key from another arbitrary length generic  
4823 secret key. It is used to produce the "master\_secret" key used in the SSL protocol from the "pre\_master"  
4824 key.

4825 It has a parameter, a **CK\_SSL3\_MASTER\_KEY\_DERIVE\_PARAMS** structure, which allows for the  
4826 passing of random data to the token. This structure is defined in Section 2.39. The *pVersion* field of the  
4827 structure must be set to NULL\_PTR since the version number is not embedded in the "pre\_master" key  
4828 as it is for RSA-like cipher suites.

4829 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
4830 key (as well as the **CKA\_VALUE\_LEN** attribute, if it is not supplied in the template). Other attributes may  
4831 be specified in the template, or else are assigned default values.

4832 The template sent along with this mechanism during a **C\_DeriveKey** call may indicate that the object  
4833 class is **CKO\_SECRET\_KEY**, the key type is **CKK\_GENERIC\_SECRET**, and the **CKA\_VALUE\_LEN**  
4834 attribute has value 48. However, since these facts are all implicit in the mechanism, there is no need to  
4835 specify any of them.

4836 This mechanism has the following rules about key sensitivity and extractability:

- 4837 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
4838 be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
4839 default value.
- 4840 • If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
4841 will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the  
4842 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
4843 **CKA\_SENSITIVE** attribute.
- 4844 • Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
4845 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
4846 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
4847 value from its **CKA\_EXTRACTABLE** attribute.

4848 For this mechanism, the ulMinKeySize and ulMaxKeySize fields of the **CK\_MECHANISM\_INFO** structure  
4849 both indicate 48 bytes.

4850 Note that this mechanism is only useable for cipher suites that do not use a fixed length 48-byte  
4851 "pre\_master" secret with an embedded version number. This includes the Diffie-Hellman cipher suites, but  
4852 excludes the RSA cipher suites.

## 4853 2.39.6 Key and MAC derivation

4854 Key, MAC and IV derivation in SSL 3.0, denoted **CKM\_SSL3\_KEY\_AND\_MAC\_DERIVE**, is a  
4855 mechanism used to derive the appropriate cryptographic keying material used by a "CipherSuite" from the  
4856 "master\_secret" key and random data. This mechanism returns the key handles for the keys generated in  
4857 the process, as well as the IVs created.

4858 It has a parameter, a **CK\_SSL3\_KEY\_MAT\_PARAMS** structure, which allows for the passing of random  
4859 data as well as the characteristic of the cryptographic material for the given CipherSuite and a pointer to a  
4860 structure which receives the handles and IVs which were generated. This structure is defined in Section  
4861 2.39.

4862 This mechanism contributes to the creation of four distinct keys on the token and returns two IVs (if IVs  
4863 are requested by the caller) back to the caller. The keys are all given an object class of  
4864 **CKO\_SECRET\_KEY**.

4865 The two MACing keys ("client\_write\_MAC\_secret" and "server\_write\_MAC\_secret") are always given a  
4866 type of **CKK\_GENERIC\_SECRET**. They are flagged as valid for signing, verification, and derivation  
4867 operations.

4868 The other two keys ("client\_write\_key" and "server\_write\_key") are typed according to information found  
4869 in the template sent along with this mechanism during a **C\_DeriveKey** function call. By default, they are  
4870 flagged as valid for encryption, decryption, and derivation operations.

4871 IVs will be generated and returned if the *ullVSizeInBits* field of the **CK\_SSL3\_KEY\_MAT\_PARAMS** field  
4872 has a nonzero value. If they are generated, their length in bits will agree with the value in the  
4873 *ullVSizeInBits* field.

4874 All four keys inherit the values of the **CKA\_SENSITIVE**, **CKA\_ALWAYS\_SENSITIVE**,  
4875 **CKA\_EXTRACTABLE**, and **CKA\_NEVER\_EXTRACTABLE** attributes from the base key. The template  
4876 provided to **C\_DeriveKey** may not specify values for any of these attributes which differ from those held  
4877 by the base key.

4878 Note that the **CK\_SSL3\_KEY\_MAT\_OUT** structure pointed to by the **CK\_SSL3\_KEY\_MAT\_PARAMS**  
 4879 structure's *pReturnedKeyMaterial* field will be modified by the **C\_DeriveKey** call. In particular, the four  
 4880 key handle fields in the **CK\_SSL3\_KEY\_MAT\_OUT** structure will be modified to hold handles to the  
 4881 newly-created keys; in addition, the buffers pointed to by the **CK\_SSL3\_KEY\_MAT\_OUT** structure's  
 4882 *pIVClient* and *pIVServer* fields will have IVs returned in them (if IVs are requested by the caller).  
 4883 Therefore, these two fields must point to buffers with sufficient space to hold any IVs that will be returned.  
 4884 This mechanism departs from the other key derivation mechanisms in Cryptoki in its returned information.  
 4885 For most key-derivation mechanisms, **C\_DeriveKey** returns a single key handle as a result of a  
 4886 successful completion. However, since the **CKM\_SSL3\_KEY\_AND\_MAC\_DERIVE** mechanism returns  
 4887 all of its key handles in the **CK\_SSL3\_KEY\_MAT\_OUT** structure pointed to by the  
 4888 **CK\_SSL3\_KEY\_MAT\_PARAMS** structure specified as the mechanism parameter, the parameter *phKey*  
 4889 passed to **C\_DeriveKey** is unnecessary, and should be a **NONE\_PTR**.  
 4890 If a call to **C\_DeriveKey** with this mechanism fails, then *none* of the four keys will be created on the  
 4891 token.

## 4892 2.39.7 MD5 MACing in SSL 3.0

4893 MD5 MACing in SSL3.0, denoted **CKM\_SSL3\_MD5\_MAC**, is a mechanism for single- and multiple-part  
 4894 signatures (data authentication) and verification using MD5, based on the SSL 3.0 protocol. This  
 4895 technique is very similar to the HMAC technique.  
 4896 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which specifies the length in bytes of the  
 4897 signatures produced by this mechanism.  
 4898 Constraints on key types and the length of input and output data are summarized in the following table:  
 4899

*Table 155, MD5 MACing in SSL 3.0: Key And Data Length*

| Function | Key type       | Data length | Signature length             |
|----------|----------------|-------------|------------------------------|
| C_Sign   | generic secret | any         | 4-8, depending on parameters |
| C_Verify | generic secret | any         | 4-8, depending on parameters |

4900 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 4901 specify the supported range of generic secret key sizes, in bits.

## 4902 2.39.8 SHA-1 MACing in SSL 3.0

4903 SHA-1 MACing in SSL3.0, denoted **CKM\_SSL3\_SHA1\_MAC**, is a mechanism for single- and multiple-part  
 4904 signatures (data authentication) and verification using SHA-1, based on the SSL 3.0 protocol. This  
 4905 technique is very similar to the HMAC technique.  
 4906 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS**, which specifies the length in bytes of the  
 4907 signatures produced by this mechanism.  
 4908 Constraints on key types and the length of input and output data are summarized in the following table:  
 4909

*Table 156, SHA-1 MACing in SSL 3.0: Key And Data Length*

| Function | Key type       | Data length | Signature length             |
|----------|----------------|-------------|------------------------------|
| C_Sign   | generic secret | any         | 4-8, depending on parameters |
| C_Verify | generic secret | any         | 4-8, depending on parameters |

4910 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 4911 specify the supported range of generic secret key sizes, in bits.

## 2.40 TLS 1.2 Mechanisms

Details for TLS 1.2 and its key derivation and MAC mechanisms can be found in [TLS12]. TLS 1.2 mechanisms differ from TLS 1.0 and 1.1 mechanisms in that the base hash used in the underlying TLS PRF (pseudo-random function) can be negotiated. Therefore each mechanism parameter for the TLS 1.2 mechanisms contains a new value in the parameters structure to specify the hash function.

This section also specifies CKM\_TLS12\_MAC which should be used in place of CKM\_TLS\_PRF to calculate the verify\_data in the TLS "finished" message.

This section also specifies CKM\_TLS\_KDF that can be used in place of CKM\_TLS\_PRF to implement key material exporters.

Table 157, TLS 1.2 Mechanisms vs. Functions

| Mechanism                      | Functions         |               |                      |        |                    |               |        |
|--------------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                                | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_TLS12_MASTER_KEY_DERIVE    |                   |               |                      |        |                    |               | ✓      |
| CKM_TLS12_MASTER_KEY_DERIVE_DH |                   |               |                      |        |                    |               | ✓      |
| CKM_TLS12_KEY_AND_MAC_DERIVE   |                   |               |                      |        |                    |               | ✓      |
| CKM_TLS12_KEY_SAFE_DERIVE      |                   |               |                      |        |                    |               | ✓      |
| CKM_TLS_KDF                    |                   |               |                      |        |                    |               | ✓      |
| CKM_TLS12_MAC                  |                   | ✓             |                      |        |                    |               |        |
| CKM_TLS12_KDF                  |                   |               |                      |        |                    |               | ✓      |

### 2.40.1 Definitions

Mechanisms:

CKM\_TLS12\_MASTER\_KEY\_DERIVE  
CKM\_TLS12\_MASTER\_KEY\_DERIVE\_DH  
CKM\_TLS12\_KEY\_AND\_MAC\_DERIVE  
CKM\_TLS12\_KEY\_SAFE\_DERIVE  
CKM\_TLS\_KDF  
CKM\_TLS12\_MAC  
CKM\_TLS12\_KDF

### 2.40.2 TLS 1.2 mechanism parameters

◆ CK\_TLS12\_MASTER\_KEY\_DERIVE\_PARAMS;  
CK\_TLS12\_MASTER\_KEY\_DERIVE\_PARAMS\_PTR

CK\_TLS12\_MASTER\_KEY\_DERIVE\_PARAMS is a structure that provides the parameters to the CKM\_TLS12\_MASTER\_KEY\_DERIVE mechanism. It is defined as follows:

```
typedef struct CK_TLS12_MASTER_KEY_DERIVE_PARAMS {
 CK_SSL3_RANDOM_DATA RandomInfo;
 CK_VERSION_PTR pVersion;
 CK_MECHANISM_TYPE prfHashMechanism;
```

```
4941 } CK_TLS12_MASTER_KEY_DERIVE_PARAMS;
4942
4943 The fields of the structure have the following meanings:
4944 RandomInfo client's and server's random data information.
4945 pVersion pointer to a CK_VERSION structure which receives the SSL
4946 protocol version information
4947 prfHashMechanism base hash used in the underlying TLS1.2 PRF operation used to
4948 derive the master key.
```

4949  
4950 **CK\_TLS12\_MASTER\_KEY\_DERIVE\_PARAMS\_PTR** is a pointer to a  
4951 **CK\_TLS12\_MASTER\_KEY\_DERIVE\_PARAMS**.

#### 4952 ♦ **CK\_TLS12\_KEY\_MAT\_PARAMS; CK\_TLS12\_KEY\_MAT\_PARAMS\_PTR**

4953 **CK\_TLS12\_KEY\_MAT\_PARAMS** is a structure that provides the parameters to the  
4954 **CKM\_TLS12\_KEY\_AND\_MAC\_DERIVE** mechanism. It is defined as follows:

```
4955 typedef struct CK_TLS12_KEY_MAT_PARAMS {
4956 CK ULONG ulMacSizeInBits;
4957 CK ULONG ulKeySizeInBits;
4958 CK ULONG ulIVSizeInBits;
4959 CK_BBOOL bIsExport;
4960 CK_SSL3_RANDOM_DATA RandomInfo;
4961 CK_SSL3_KEY_MAT_OUT_PTR pReturnedKeyMaterial;
4962 CK_MECHANISM_TYPE prfHashMechanism;
4963 } CK_TLS12_KEY_MAT_PARAMS;
```

4964  
4965 The fields of the structure have the following meanings:

4966 ulMacSizeInBits the length (in bits) of the MACing keys agreed upon during the  
4967 protocol handshake phase. If no MAC key is required, the length  
4968 should be set to 0.  
4969 ulKeySizeInBits the length (in bits) of the secret keys agreed upon during the  
4970 protocol handshake phase  
4971 ulIVSizeInBits the length (in bits) of the IV agreed upon during the protocol  
4972 handshake phase. If no IV is required, the length should be set to 0  
4973 bIsExport must be set to CK\_FALSE because export cipher suites must not be  
4974 used in TLS 1.1 and later.  
4975 RandomInfo client's and server's random data information.  
4976 pReturnedKeyMaterial points to a CK\_SSL3\_KEY\_MAT\_OUT structures which receives  
4977 the handles for the keys generated and the IVs  
4978 prfHashMechanism base hash used in the underlying TLS1.2 PRF operation used to  
4979 derive the master key.

4980   **CK\_TLS12\_KEY\_MAT\_PARAMS\_PTR** is a pointer to a **CK\_TLS12\_KEY\_MAT\_PARAMS**.

4981   ◆ **CK\_TLS\_KDF\_PARAMS; CK\_TLS\_KDF\_PARAMS\_PTR**

4982   **CK\_TLS\_KDF\_PARAMS** is a structure that provides the parameters to the CKM\_TLS\_KDF mechanism.

4983   It is defined as follows:

```

4984 typedef struct CK_TLS_KDF_PARAMS {
4985 CK_MECHANISM_TYPE prfMechanism;
4986 CK_BYTE_PTR pLabel;
4987 CK ULONG ulLabelLength;
4988 CK_SSL3_RANDOM_DATA RandomInfo;
4989 CK_BYTE_PTR pContextData;
4990 CK ULONG ulContextDataLength;
4991 } CK_TLS_KDF_PARAMS;

```

4992

4993   The fields of the structure have the following meanings:

|                                 |                                                                                                                         |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| 4994 <i>prfMechanism</i>        | <i>the hash mechanism used in the TLS1.2 PRF construct or CKM_TLS_PRF to use with the TLS1.0 and 1.1 PRF construct.</i> |
| 4996 <i>pLabel</i>              | <i>a pointer to the label for this key derivation</i>                                                                   |
| 4997 <i>ulLabelLength</i>       | <i>length of the label in bytes</i>                                                                                     |
| 4998 <i>RandomInfo</i>          | <i>the random data for the key derivation</i>                                                                           |
| 4999 <i>pContextData</i>        | <i>a pointer to the context data for this key derivation. NULL_PTR if not present</i>                                   |
| 5001 <i>ulContextDataLength</i> | <i>length of the context data in bytes. 0 if not present.</i>                                                           |

5002   **CK\_TLS\_KDF\_PARAMS\_PTR** is a pointer to a **CK\_TLS\_KDF\_PARAMS**.

5003   ◆ **CK\_TLS\_MAC\_PARAMS; CK\_TLS\_MAC\_PARAMS\_PTR**

5004   **CK\_TLS\_MAC\_PARAMS** is a structure that provides the parameters to the **CKM\_TLS\_MAC**

5005   mechanism. It is defined as follows:

```

5006 typedef struct CK_TLS_MAC_PARAMS {
5007 CK_MECHANISM_TYPE prfMechanism;
5008 CK ULONG ulMacLength;
5009 CK ULONG ulServerOrClient;
5010 } CK_TLS_MAC_PARAMS;

```

5011

5012   The fields of the structure have the following meanings:

|                          |                                                                                                                                                                    |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5013 <i>prfMechanism</i> | <i>the hash mechanism used in the TLS12 PRF construct or CKM_TLS_PRF to use with the TLS1.0 and 1.1 PRF construct.</i>                                             |
| 5015 <i>ulMacLength</i>  | <i>the length of the MAC tag required or offered. Always 12 octets in TLS 1.0 and 1.1. Generally 12 octets, but may be negotiated to a longer value in TLS1.2.</i> |

5018                   *ulServerOrClient*     1 to use the label "server finished", 2 to use the label "client  
5019                   finished". All other values are invalid.

5020   **CK\_TLS\_MAC\_PARAMS\_PTR** is a pointer to a **CK\_TLS\_MAC\_PARAMS**.

5021

5022   ◆ **CK\_TLS\_PRF\_PARAMS; CK\_TLS\_PRF\_PARAMS\_PTR**

5023   **CK\_TLS\_PRF\_PARAMS** is a structure, which provides the parameters to the **CKM\_TLS\_PRF**  
5024 mechanism. It is defined as follows:

```
5025 typedef struct CK_TLS_PRF_PARAMS {
5026 CK_BYTE_PTR pSeed;
5027 CK ULONG ulSeedLen;
5028 CK_BYTE_PTR pLabel;
5029 CK ULONG ulLabelLen;
5030 CK_BYTE_PTR pOutput;
5031 CK ULONG_PTR pulOutputLen;
5032 } CK_TLS_PRF_PARAMS;
```

5033

5034   The fields of the structure have the following meanings:

5035                   *pSeed*       pointer to the input seed

5036                   *ulSeedLen*   length in bytes of the input seed

5037                   *pLabel*      pointer to the identifying label

5038                   *ulLabelLen*   length in bytes of the identifying label

5039                   *pOutput*     pointer receiving the output of the operation

5040                   *pulOutputLen*   pointer to the length in bytes that the output to be created shall  
5041                   have, has to hold the desired length as input and will receive the  
5042                   calculated length as output

5043   **CK\_TLS\_PRF\_PARAMS\_PTR** is a pointer to a **CK\_TLS\_PRF\_PARAMS**.

## 5044   **2.40.3 TLS MAC**

5045   The TLS MAC mechanism is used to generate integrity tags for the TLS "finished" message. It replaces  
5046   the use of the **CKM\_TLS\_PRF** function for TLS1.0 and 1.1 and that mechanism is deprecated.

5047   **CKM\_TLS\_MAC** takes a parameter of **CK\_TLS\_MAC\_PARAMS**. To use this mechanism with TLS1.0  
5048   and TLS1.1, use **CKM\_TLS\_PRF** as the value for *prfMechanism* in place of a hash mechanism. Note:  
5049   Although **CKM\_TLS\_PRF** is deprecated as a mechanism for **C\_DeriveKey**, the manifest value is retained  
5050   for use with this mechanism to indicate the use of the TLS1.0/1.1 pseudo-random function.

5051   In TLS1.0 and 1.1 the "finished" message *verify\_data* (i.e. the output signature from the MAC mechanism)  
5052   is always 12 bytes. In TLS1.2 the "finished" message *verify\_data* is a minimum of 12 bytes, defaults to 12  
5053   bytes, but may be negotiated to longer length.

5054 Table 158, General-length TLS MAC: Key And Data Length

| Function | Key type       | Data length | Signature length |
|----------|----------------|-------------|------------------|
| C_Sign   | generic secret | any         | >=12 bytes       |
| C_Verify | generic secret | any         | >=12 bytes       |

5055

#### 5056 2.40.4 Master key derivation

5057 Master key derivation in TLS 1.0, denoted **CKM\_TLS\_MASTER\_KEY\_DERIVE**, is a mechanism used to  
5058 derive one 48-byte generic secret key from another 48-byte generic secret key. It is used to produce the  
5059 "master\_secret" key used in the TLS protocol from the "pre\_master" key. This mechanism returns the  
5060 value of the client version, which is built into the "pre\_master" key as well as a handle to the derived  
5061 "master\_secret" key.

5062 It has a parameter, a **CK\_SSL3\_MASTER\_KEY\_DERIVE\_PARAMS** structure, which allows for the  
5063 passing of random data to the token as well as the returning of the protocol version number which is part  
5064 of the pre-master key. This structure is defined in Section 2.39.

5065 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
5066 key (as well as the **CKA\_VALUE\_LEN** attribute, if it is not supplied in the template). Other attributes may  
5067 be specified in the template, or else are assigned default values.

5068 The mechanism also contributes the **CKA\_ALLOWED\_MECHANISMS** attribute consisting only of  
5069 **CKM\_TLS12\_KEY\_AND\_MAC\_DERIVE**, **CKM\_TLS12\_KEY\_SAFE\_DERIVE**, **CKM\_TLS12\_KDF** and  
5070 **CKM\_TLS12\_MAC**.

5071 The template sent along with this mechanism during a **C\_DeriveKey** call may indicate that the object  
5072 class is **CKO\_SECRET\_KEY**, the key type is **CKK\_GENERIC\_SECRET**, and the **CKA\_VALUE\_LEN**  
5073 attribute has value 48. However, since these facts are all implicit in the mechanism, there is no need to  
5074 specify any of them.

5075 This mechanism has the following rules about key sensitivity and extractability:

- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
5077 be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
5078 default value.
- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
5079 will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the  
5080 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
5081 **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
5083 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
5084 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
5085 value from its **CKA\_EXTRACTABLE** attribute.

5087 For this mechanism, the ulMinKeySize and ulMaxKeySize fields of the **CK\_MECHANISM\_INFO** structure  
5088 both indicate 48 bytes.

5089 Note that the **CK\_VERSION** structure pointed to by the **CK\_SSL3\_MASTER\_KEY\_DERIVE\_PARAMS**  
5090 structure's *pVersion* field will be modified by the **C\_DeriveKey** call. In particular, when the call returns,  
5091 this structure will hold the SSL version associated with the supplied pre\_master key.

5092 Note that this mechanism is only useable for cipher suites that use a 48-byte "pre\_master" secret with an  
5093 embedded version number. This includes the RSA cipher suites, but excludes the Diffie-Hellman cipher  
5094 suites.

#### 5095 2.40.5 Master key derivation for Diffie-Hellman

5096 Master key derivation for Diffie-Hellman in TLS 1.0, denoted **CKM\_TLS\_MASTER\_KEY\_DERIVE\_DH**, is  
5097 a mechanism used to derive one 48-byte generic secret key from another arbitrary length generic secret  
5098 key. It is used to produce the "master\_secret" key used in the TLS protocol from the "pre\_master" key.

5099 It has a parameter, a **CK\_SSL3\_MASTER\_KEY\_DERIVE\_PARAMS** structure, which allows for the  
5100 passing of random data to the token. This structure is defined in Section 2.39. The *pVersion* field of the  
5101 structure must be set to **NULL\_PTR** since the version number is not embedded in the "pre\_master" key  
5102 as it is for RSA-like cipher suites.

5103 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
5104 key (as well as the **CKA\_VALUE\_LEN** attribute, if it is not supplied in the template). Other attributes may  
5105 be specified in the template, or else are assigned default values.

5106 The mechanism also contributes the **CKA\_ALLOWED\_MECHANISMS** attribute consisting only of  
5107 **CKM\_TLS12\_KEY\_AND\_MAC\_DERIVE**, **CKM\_TLS12\_KEY\_SAFE\_DERIVE**, **CKM\_TLS12\_KDF** and  
5108 **CKM\_TLS12\_MAC**.

5109 The template sent along with this mechanism during a **C\_DeriveKey** call may indicate that the object  
5110 class is **CKO\_SECRET\_KEY**, the key type is **CKK\_GENERIC\_SECRET**, and the **CKA\_VALUE\_LEN**  
5111 attribute has value 48. However, since these facts are all implicit in the mechanism, there is no need to  
5112 specify any of them.

5113 This mechanism has the following rules about key sensitivity and extractability:

- 5114 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
5115 be specified to be either **CK\_TRUE** or **CK\_FALSE**. If omitted, these attributes each take on some  
5116 default value.
- 5117 • If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to **CK\_FALSE**, then the derived key  
5118 will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to **CK\_TRUE**, then the  
5119 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
5120 **CKA\_SENSITIVE** attribute.
- 5121 • Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to **CK\_FALSE**, then the  
5122 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
5123 **CK\_TRUE**, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
5124 value from its **CKA\_EXTRACTABLE** attribute.

5125 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
5126 both indicate 48 bytes.

5127 Note that this mechanism is only useable for cipher suites that do not use a fixed length 48-byte  
5128 "pre\_master" secret with an embedded version number. This includes the Diffie-Hellman cipher suites, but  
5129 excludes the RSA cipher suites.

## 5130 **2.40.6 Key and MAC derivation**

5131 Key, MAC and IV derivation in TLS 1.0, denoted **CKM\_TLS\_KEY\_AND\_MAC\_DERIVE**, is a mechanism  
5132 used to derive the appropriate cryptographic keying material used by a "CipherSuite" from the  
5133 "master\_secret" key and random data. This mechanism returns the key handles for the keys generated in  
5134 the process, as well as the IVs created.

5135 It has a parameter, a **CK\_SSL3\_KEY\_MAT\_PARAMS** structure, which allows for the passing of random  
5136 data as well as the characteristic of the cryptographic material for the given CipherSuite and a pointer to a  
5137 structure which receives the handles and IVs which were generated. This structure is defined in Section  
5138 2.39.

5139 This mechanism contributes to the creation of four distinct keys on the token and returns two IVs (if IVs  
5140 are requested by the caller) back to the caller. The keys are all given an object class of  
5141 **CKO\_SECRET\_KEY**.

5142 The two MACing keys ("client\_write\_MAC\_secret" and "server\_write\_MAC\_secret") (if present) are  
5143 always given a type of **CKK\_GENERIC\_SECRET**. They are flagged as valid for signing and verification.

5144 The other two keys ("client\_write\_key" and "server\_write\_key") are typed according to information found  
5145 in the template sent along with this mechanism during a **C\_DeriveKey** function call. By default, they are  
5146 flagged as valid for encryption, decryption, and derivation operations.

5147 For **CKM\_TLS12\_KEY\_AND\_MAC\_DERIVE**, IVs will be generated and returned if the *ullVSizeInBits*  
5148 field of the **CK\_SSL3\_KEY\_MAT\_PARAMS** field has a nonzero value. If they are generated, their length  
5149 in bits will agree with the value in the *ullVSizeInBits* field.

5150

5151 Note Well: CKM\_TLS12\_KEY\_AND\_MAC\_DERIVE produces both private (key) and public (IV)  
5152 data. It is possible to "leak" private data by the simple expedient of decreasing the length of  
5153 private data requested. E.g. Setting *ulMacSizeInBits* and *ulKeySizeInBits* to 0 (or other lengths  
5154 less than the key size) will result in the private key data being placed in the destination  
5155 designated for the IV's. Repeated calls with the same master key and same RandomInfo but with  
5156 differing lengths for the private key material will result in different data being leaked.<

5157

5158 All four keys inherit the values of the **CKA\_SENSITIVE**, **CKA\_ALWAYS\_SENSITIVE**,  
5159 **CKA\_EXTRACTABLE**, and **CKA\_NEVER\_EXTRACTABLE** attributes from the base key. The template  
5160 provided to **C\_DeriveKey** may not specify values for any of these attributes which differ from those held  
5161 by the base key.

5162 Note that the **CK\_SSL3\_KEY\_MAT\_OUT** structure pointed to by the **CK\_SSL3\_KEY\_MAT\_PARAMS**  
5163 structure's *pReturnedKeyMaterial* field will be modified by the **C\_DeriveKey** call. In particular, the four  
5164 key handle fields in the **CK\_SSL3\_KEY\_MAT\_OUT** structure will be modified to hold handles to the  
5165 newly-created keys; in addition, the buffers pointed to by the **CK\_SSL3\_KEY\_MAT\_OUT** structure's  
5166 *pIVClient* and *pIVServer* fields will have IVs returned in them (if IVs are requested by the caller).  
5167 Therefore, these two fields must point to buffers with sufficient space to hold any IVs that will be returned.

5168 This mechanism departs from the other key derivation mechanisms in Cryptoki in its returned information.  
5169 For most key-derivation mechanisms, **C\_DeriveKey** returns a single key handle as a result of a  
5170 successful completion. However, since the **CKM\_TLS12\_KEY\_AND\_MAC\_DERIVE** mechanism returns  
5171 all of its key handles in the **CK\_SSL3\_KEY\_MAT\_OUT** structure pointed to by the  
5172 **CK\_SSL3\_KEY\_MAT\_PARAMS** structure specified as the mechanism parameter, the parameter *phKey*  
5173 passed to **C\_DeriveKey** is unnecessary, and should be a **NONE\_PTR**.

5174 If a call to **C\_DeriveKey** with this mechanism fails, then *none* of the four keys will be created on the  
5175 token.

## 5176 **2.40.7 CKM\_TLS12\_KEY\_SAFE\_DERIVE**

5177 **CKM\_TLS12\_KEY\_SAFE\_DERIVE** is identical to **CKM\_TLS12\_KEY\_AND\_MAC\_DERIVE** except that it  
5178 shall never produce IV data, and the *ullvSizeInBits* field of **CK\_TLS12\_KEY\_MAT\_PARAMS** is ignored  
5179 and treated as 0. All of the other conditions and behavior described for  
5180 **CKM\_TLS12\_KEY\_AND\_MAC\_DERIVE**, with the exception of the black box warning, apply to this  
5181 mechanism.

5182 **CKM\_TLS12\_KEY\_SAFE\_DERIVE** is provided as a separate mechanism to allow a client to control the  
5183 export of IV material (and possible leaking of key material) through the use of the  
5184 **CKA\_ALLOWED\_MECHANISMS** key attribute.

## 5185 **2.40.8 Generic Key Derivation using the TLS PRF**

5186 **CKM\_TLS\_KDF** is the mechanism defined in [RFC 5705]. It uses the TLS key material and TLS PRF  
5187 function to produce additional key material for protocols that want to leverage the TLS key negotiation  
5188 mechanism. **CKM\_TLS\_KDF** has a parameter of **CK\_TLS\_KDF\_PARAMS**. If the protocol using this  
5189 mechanism does not use context information, the *pContextData* field shall be set to **NONE\_PTR** and the  
5190 *ulContextDataLength* field shall be set to 0.

5191 To use this mechanism with TLS1.0 and TLS1.1, use **CKM\_TLS\_PRF** as the value for *prfMechanism* in  
5192 place of a hash mechanism. Note: Although **CKM\_TLS\_PRF** is deprecated as a mechanism for  
5193 **C\_DeriveKey**, the manifest value is retained for use with this mechanism to indicate the use of the  
5194 TLS1.0/1.1 Pseudo-random function.

5195 This mechanism can be used to derive multiple keys (e.g. similar to  
5196 **CKM\_TLS12\_KEY\_AND\_MAC\_DERIVE**) by first deriving the key stream as a **CKK\_GENERIC\_SECRET**  
5197 of the necessary length and doing subsequent derives against that derived key using the  
5198 **CKM\_EXTRACT\_KEY\_FROM\_KEY** mechanism to split the key stream into the actual operational keys.

5199 The mechanism should not be used with the labels defined for use with TLS, but the token does not  
5200 enforce this behavior.

5201 This mechanism has the following rules about key sensitivity and extractability:

- 5202 • If the original key has its **CKA\_SENSITIVE** attribute set to CK\_TRUE, so does the derived key. If not,  
5203 then the derived key's **CKA\_SENSITIVE** attribute is set either from the supplied template or from the  
5204 original key.
- 5205 • Similarly, if the original key has its **CKA\_EXTRACTABLE** attribute set to CK\_FALSE, so does the  
5206 derived key. If not, then the derived key's **CKA\_EXTRACTABLE** attribute is set either from the  
5207 supplied template or from the original key.
- 5208 • The derived key's **CKA\_ALWAYS\_SENSITIVE** attribute is set to CK\_TRUE if and only if the original  
5209 key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE.
- 5210 • Similarly, the derived key's **CKA\_NEVER\_EXTRACTABLE** attribute is set to CK\_TRUE if and only if  
5211 the original key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE.

## 5212 **2.40.9 Generic Key Derivation using the TLS12 PRF**

5213 **CKM\_TLS12\_KDF** is the mechanism defined in [RFC 5705]. It uses the TLS key material and TLS PRF  
5214 function to produce additional key material for protocols that want to leverage the TLS key negotiation  
5215 mechanism. **CKM\_TLS12\_KDF** has a parameter of **CK\_TLS\_KDF\_PARAMS**. If the protocol using this  
5216 mechanism does not use context information, the *pContextData* field shall be set to *NULL\_PTR* and the  
5217 *ulContextDataLength* field shall be set to 0.

5218 To use this mechanism with TLS1.0 and TLS1.1, use **CKM\_TLS\_PRF** as the value for *prfMechanism* in  
5219 place of a hash mechanism. Note: Although **CKM\_TLS\_PRF** is deprecated as a mechanism for  
5220 *C\_DeriveKey*, the manifest value is retained for use with this mechanism to indicate the use of the  
5221 TLS1.0/1.1 Pseudo-random function.

5222 This mechanism can be used to derive multiple keys (e.g. similar to  
5223 **CKM\_TLS12\_KEY\_AND\_MAC\_DERIVE**) by first deriving the key stream as a **CKK\_GENERIC\_SECRET**  
5224 of the necessary length and doing subsequent derives against that derived key stream using the  
5225 **CKM\_EXTRACT\_KEY\_FROM\_KEY** mechanism to split the key stream into the actual operational keys.

5226 The mechanism should not be used with the labels defined for use with TLS, but the token does not  
5227 enforce this behavior.

5228 This mechanism has the following rules about key sensitivity and extractability:

- 5229 • If the original key has its **CKA\_SENSITIVE** attribute set to CK\_TRUE, so does the derived key. If not,  
5230 then the derived key's **CKA\_SENSITIVE** attribute is set either from the supplied template or from the  
5231 original key.
- 5232 • Similarly, if the original key has its **CKA\_EXTRACTABLE** attribute set to CK\_FALSE, so does the  
5233 derived key. If not, then the derived key's **CKA\_EXTRACTABLE** attribute is set either from the  
5234 supplied template or from the original key.
- 5235 • The derived key's **CKA\_ALWAYS\_SENSITIVE** attribute is set to CK\_TRUE if and only if the original  
5236 key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE.
- 5237 • Similarly, the derived key's **CKA\_NEVER\_EXTRACTABLE** attribute is set to CK\_TRUE if and only if  
5238 the original key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE.

## 2.41 WTLS

Details can be found in [WTLS].

When comparing the existing TLS mechanisms with these extensions to support WTLS one could argue that there would be no need to have distinct handling of the client and server side of the handshake. However, since in WTLS the server and client use different sequence numbers, there could be instances (e.g. when WTLS is used to protect asynchronous protocols) where sequence numbers on the client and server side differ, and hence this motivates the introduced split.

Table 159, WTLS Mechanisms vs. Functions

| Mechanism                          | Functions         |               |                      |        |                         |               |        |
|------------------------------------|-------------------|---------------|----------------------|--------|-------------------------|---------------|--------|
|                                    | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Generate Key / Key Pair | Wrap & Unwrap | Derive |
| CKM_WTLS_PRE_MASTER_KEY_GEN        |                   |               |                      |        | ✓                       |               |        |
| CKM_WTLS_MASTER_KEY_DERIVE         |                   |               |                      |        |                         |               | ✓      |
| CKM_WTLS_MASTER_KEY_DERIVE_DH_ECC  |                   |               |                      |        |                         |               | ✓      |
| CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE |                   |               |                      |        |                         |               | ✓      |
| CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE |                   |               |                      |        |                         |               | ✓      |
| CKM_WTLS_PRF                       |                   |               |                      |        |                         |               | ✓      |

### 2.41.1 Definitions

Mechanisms:

CKM\_WTLS\_PRE\_MASTER\_KEY\_GEN  
CKM\_WTLS\_MASTER\_KEY\_DERIVE  
CKM\_WTLS\_MASTER\_KEY\_DERIVE\_DH\_ECC  
CKM\_WTLS\_PRF  
CKM\_WTLS\_SERVER\_KEY\_AND\_MAC\_DERIVE  
CKM\_WTLS\_CLIENT\_KEY\_AND\_MAC\_DERIVE

### 2.41.2 WTLS mechanism parameters

#### ◆ CK\_WTLS\_RANDOM\_DATA; CK\_WTLS\_RANDOM\_DATA\_PTR

**CK\_WTLS\_RANDOM\_DATA** is a structure, which provides information about the random data of a client and a server in a WTLS context. This structure is used by the **CKM\_WTLS\_MASTER\_KEY\_DERIVE** mechanism. It is defined as follows:

```
typedef struct CK_WTLS_RANDOM_DATA {
 CK_BYTE_PTR pClientRandom;
 CK ULONG ulClientRandomLen;
 CK_BYTE_PTR pServerRandom;
```

```

5265 CK ULONG ulServerRandomLen;
5266 } CK_WTLS_RANDOM_DATA;
5267
5268 The fields of the structure have the following meanings:
5269 pClientRandom pointer to the client's random data
5270 pClientRandomLen length in bytes of the client's random data
5271 pServerRaandom pointer to the server's random data
5272 ulServerRandomLen length in bytes of the server's random data
5273 CK_WTLS_RANDOM_DATA_PTR is a pointer to a CK_WTLS_RANDOM_DATA.
5274 ♦ CK_WTLS_MASTER_KEY_DERIVE_PARAMS;
5275 CK_WTLS_MASTER_KEY_DERIVE_PARAMS_PTR
5276 CK_WTLS_MASTER_KEY_DERIVE_PARAMS is a structure, which provides the parameters to the
5277 CKM_WTLS_MASTER_KEY_DERIVE mechanism. It is defined as follows:
5278 typedef struct CK_WTLS_MASTER_KEY_DERIVE_PARAMS {
5279 CK_MECHANISM_TYPE DigestMechanism;
5280 CK_WTLS_RANDOM_DATA RandomInfo;
5281 CK_BYTE_PTR pVersion;
5282 } CK_WTLS_MASTER_KEY_DERIVE_PARAMS;
5283
5284 The fields of the structure have the following meanings:
5285 DigestMechanism the mechanism type of the digest mechanism to be used (possible
5286 types can be found in [WTLS])
5287 RandomInfo Client's and server's random data information
5288 pVersion pointer to a CK_BYTE which receives the WTLS protocol version
5289 information
5290 CK_WTLS_MASTER_KEY_DERIVE_PARAMS_PTR is a pointer to a
5291 CK_WTLS_MASTER_KEY_DERIVE_PARAMS.
5292 ♦ CK_WTLS_PRF_PARAMS; CK_WTLS_PRF_PARAMS_PTR
5293 CK_WTLS_PRF_PARAMS is a structure, which provides the parameters to the CKM_WTLS_PRF
5294 mechanism. It is defined as follows:
5295 typedef struct CK_WTLS_PRF_PARAMS {
5296 CK_MECHANISM_TYPE DigestMechanism;
5297 CK_BYTE_PTR pSeed;
5298 CK ULONG ulSeedLen;
5299 CK_BYTE_PTR pLabel;
5300 CK ULONG ulLabelLen;
5301 CK_BYTE_PTR pOutput;
5302 CK ULONG_PTR pulOutputLen;
5303 } CK_WTLS_PRF_PARAMS;

```

5304

5305 The fields of the structure have the following meanings:

|      |                         |                                                                                                                                                                          |
|------|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5306 | <i>Digest Mechanism</i> | <i>the mechanism type of the digest mechanism to be used (possible types can be found in [WTLS])</i>                                                                     |
| 5307 |                         |                                                                                                                                                                          |
| 5308 | <i>pSeed</i>            | <i>pointer to the input seed</i>                                                                                                                                         |
| 5309 | <i>ulSeedLen</i>        | <i>length in bytes of the input seed</i>                                                                                                                                 |
| 5310 | <i>pLabel</i>           | <i>pointer to the identifying label</i>                                                                                                                                  |
| 5311 | <i>ulLabelLen</i>       | <i>length in bytes of the identifying label</i>                                                                                                                          |
| 5312 | <i>pOutput</i>          | <i>pointer receiving the output of the operation</i>                                                                                                                     |
| 5313 | <i>pulOutputLen</i>     | <i>pointer to the length in bytes that the output to be created shall have, has to hold the desired length as input and will receive the calculated length as output</i> |
| 5314 |                         |                                                                                                                                                                          |
| 5315 |                         |                                                                                                                                                                          |

5316 **CK\_WTLS\_PRF\_PARAMS\_PTR** is a pointer to a **CK\_WTLS\_PRF\_PARAMS**.

5317 ◆ **CK\_WTLS\_KEY\_MAT\_OUT; CK\_WTLS\_KEY\_MAT\_OUT\_PTR**

5318 **CK\_WTLS\_KEY\_MAT\_OUT** is a structure that contains the resulting key handles and initialization vectors after performing a C\_DeriveKey function with the  
 5319 **CKM\_WTLS\_SERVER\_KEY\_AND\_MAC\_DERIVE** or with the  
 5320 **CKM\_WTLS\_CLIENT\_KEY\_AND\_MAC\_DERIVE** mechanism. It is defined as follows:

```
5322 typedef struct CK_WTLS_KEY_MAT_OUT {
5323 CK_OBJECT_HANDLE hMacSecret;
5324 CK_OBJECT_HANDLE hKey;
5325 CK_BYTE_PTR pIV;
5326 } CK_WTLS_KEY_MAT_OUT;
```

5327

5328 The fields of the structure have the following meanings:

|      |                   |                                                                                             |
|------|-------------------|---------------------------------------------------------------------------------------------|
| 5329 | <i>hMacSecret</i> | <i>Key handle for the resulting MAC secret key</i>                                          |
| 5330 | <i>hKey</i>       | <i>Key handle for the resulting secret key</i>                                              |
| 5331 | <i>pIV</i>        | <i>Pointer to a location which receives the initialization vector (IV) created (if any)</i> |
| 5332 |                   |                                                                                             |

5333 **CK\_WTLS\_KEY\_MAT\_OUT\_PTR** is a pointer to a **CK\_WTLS\_KEY\_MAT\_OUT**.

5334 ◆ **CK\_WTLS\_KEY\_MAT\_PARAMS; CK\_WTLS\_KEY\_MAT\_PARAMS\_PTR**

5335 **CK\_WTLS\_KEY\_MAT\_PARAMS** is a structure that provides the parameters to the  
 5336 **CKM\_WTLS\_SERVER\_KEY\_AND\_MAC\_DERIVE** and the  
 5337 **CKM\_WTLS\_CLIENT\_KEY\_AND\_MAC\_DERIVE** mechanisms. It is defined as follows:

```
5338 typedef struct CK_WTLS_KEY_MAT_PARAMS {
5339 CK_MECHANISM_TYPE DigestMechanism;
5340 CK ULONG ulMacSizeInBits;
5341 CK ULONG ulKeySizeInBits;
```

```

5342 CK ULONG ulIVSizeInBits;
5343 CK ULONG ulSequenceNumber;
5344 CK_BBOOL bIsExport;
5345 CK_WTLS_RANDOM_DATA RandomInfo;
5346 CK_WTLS_KEY_MAT_OUT_PTR pReturnedKeyMaterial;
5347 } CK_WTLS_KEY_MAT_PARAMS;

```

5348

5349 The fields of the structure have the following meanings:

|      |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5350 | <i>Digest Mechanism</i>     | <i>the mechanism type of the digest mechanism to be used (possible types can be found in [WTLS])</i>                                                                                                                                                                                                                                                                                                                                                                                                 |
| 5351 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 5352 | <i>ulMaxSizeInBits</i>      | <i>the length (in bits) of the MACing key agreed upon during the protocol handshake phase</i>                                                                                                                                                                                                                                                                                                                                                                                                        |
| 5353 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 5354 | <i>ulKeySizeInBits</i>      | <i>the length (in bits) of the secret key agreed upon during the handshake phase</i>                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 5355 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 5356 | <i>ullIVSizeInBits</i>      | <i>the length (in bits) of the IV agreed upon during the handshake phase. If no IV is required, the length should be set to 0.</i>                                                                                                                                                                                                                                                                                                                                                                   |
| 5357 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 5358 | <i>ulSequenceNumber</i>     | <i>the current sequence number used for records sent by the client and server respectively</i>                                                                                                                                                                                                                                                                                                                                                                                                       |
| 5359 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 5360 | <i>bIsExport</i>            | <i>a boolean value which indicates whether the keys have to be derived for an export version of the protocol. If this value is true (i.e., the keys are exportable) then <i>ulKeySizeInBits</i> is the length of the key in bits before expansion. The length of the key after expansion is determined by the information found in the template sent along with this mechanism during a <i>C_DeriveKey</i> function call (either the <b>CKA_KEY_TYPE</b> or the <b>CKA_VALUE_LEN</b> attribute).</i> |
| 5361 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 5362 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 5363 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 5364 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 5365 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 5366 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 5367 | <i>RandomInfo</i>           | <i>client's and server's random data information</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 5368 | <i>pReturnedKeyMaterial</i> | <i>points to a <b>CK_WTLS_KEY_MAT_OUT</b> structure which receives the handles for the keys generated and the IV</i>                                                                                                                                                                                                                                                                                                                                                                                 |
| 5369 |                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

5370 **CK\_WTLS\_KEY\_MAT\_PARAMS\_PTR** is a pointer to a **CK\_WTLS\_KEY\_MAT\_PARAMS**.

### 5371 **2.41.3 Pre master secret key generation for RSA key exchange suite**

5372 Pre master secret key generation for the RSA key exchange suite in WTLS denoted  
**CKM\_WTLS\_PRE\_MASTER\_KEY\_GEN**, is a mechanism, which generates a variable length secret key.  
5373 It is used to produce the pre master secret key for RSA key exchange suite used in WTLS. This  
5375 mechanism returns a handle to the pre master secret key.

5376 It has one parameter, a **CK\_BYTE**, which provides the client's WTLS version.

5377 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE** and **CKA\_VALUE** attributes to the new  
5378 key (as well as the **CKA\_VALUE\_LEN** attribute, if it is not supplied in the template). Other attributes may  
5379 be specified in the template, or else are assigned default values.

5380 The template sent along with this mechanism during a **C\_GenerateKey** call may indicate that the object  
5381 class is **CKO\_SECRET\_KEY**, the key type is **CKK\_GENERIC\_SECRET**, and the **CKA\_VALUE\_LEN**  
5382 attribute indicates the length of the pre master secret key.

5383 For this mechanism, the *ulMinKeySize* field of the **CK\_MECHANISM\_INFO** structure shall indicate 20  
5384 bytes.

#### 5385    2.41.4 Master secret key derivation

5386    Master secret derivation in WTLS, denoted **CKM\_WTLS\_MASTER\_KEY\_DERIVE**, is a mechanism used  
5387    to derive a 20 byte generic secret key from variable length secret key. It is used to produce the master  
5388    secret key used in WTLS from the pre master secret key. This mechanism returns the value of the client  
5389    version, which is built into the pre master secret key as well as a handle to the derived master secret key.

5390    It has a parameter, a **CK\_WTLS\_MASTER\_KEY\_DERIVE\_PARAMS** structure, which allows for passing  
5391    the mechanism type of the digest mechanism to be used as well as the passing of random data to the  
5392    token as well as the returning of the protocol version number which is part of the pre master secret key.

5393    The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
5394    key (as well as the **CKA\_VALUE\_LEN** attribute, if it is not supplied in the template). Other attributes may  
5395    be specified in the template, or else are assigned default values.

5396    The template sent along with this mechanism during a **C\_DeriveKey** call may indicate that the object  
5397    class is **CKO\_SECRET\_KEY**, the key type is **CKK\_GENERIC\_SECRET**, and the **CKA\_VALUE\_LEN**  
5398    attribute has value 20. However, since these facts are all implicit in the mechanism, there is no need to  
5399    specify any of them.

5400    This mechanism has the following rules about key sensitivity and extractability:

5401    The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both be  
5402    specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some default  
5403    value.

5404    If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key will  
5405    as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the derived  
5406    key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its **CKA\_SENSITIVE** attribute.

5407    Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
5408    derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE,  
5409    then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its  
5410    **CKA\_EXTRACTABLE** attribute.

5411    For this mechanism, the ulMinKeySize and ulMaxKeySize fields of the **CK\_MECHANISM\_INFO** structure  
5412    both indicate 20 bytes.

5413    Note that the **CK\_BYTE** pointed to by the **CK\_WTLS\_MASTER\_KEY\_DERIVE\_PARAMS** structure's  
5414    *pVersion* field will be modified by the **C\_DeriveKey** call. In particular, when the call returns, this byte will  
5415    hold the WTLS version associated with the supplied pre master secret key.

5416    Note that this mechanism is only useable for key exchange suites that use a 20-byte pre master secret  
5417    key with an embedded version number. This includes the RSA key exchange suites, but excludes the  
5418    Diffie-Hellman and Elliptic Curve Cryptography key exchange suites.

#### 5419    2.41.5 Master secret key derivation for Diffie-Hellman and Elliptic Curve 5420    Cryptography

5421    Master secret derivation for Diffie-Hellman and Elliptic Curve Cryptography in WTLS, denoted  
5422    **CKM\_WTLS\_MASTER\_KEY\_DERIVE\_DH\_ECC**, is a mechanism used to derive a 20 byte generic  
5423    secret key from variable length secret key. It is used to produce the master secret key used in WTLS from  
5424    the pre master secret key. This mechanism returns a handle to the derived master secret key.

5425    It has a parameter, a **CK\_WTLS\_MASTER\_KEY\_DERIVE\_PARAMS** structure, which allows for the  
5426    passing of the mechanism type of the digest mechanism to be used as well as random data to the token.  
5427    The *pVersion* field of the structure must be set to NULL\_PTR since the version number is not embedded  
5428    in the pre master secret key as it is for RSA-like key exchange suites.

5429    The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
5430    key (as well as the **CKA\_VALUE\_LEN** attribute, if it is not supplied in the template). Other attributes may  
5431    be specified in the template, or else are assigned default values.

5432    The template sent along with this mechanism during a **C\_DeriveKey** call may indicate that the object  
5433    class is **CKO\_SECRET\_KEY**, the key type is **CKK\_GENERIC\_SECRET**, and the **CKA\_VALUE\_LEN**

5434 attribute has value 20. However, since these facts are all implicit in the mechanism, there is no need to  
5435 specify any of them.

5436 This mechanism has the following rules about key sensitivity and extractability:

5437 The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both be  
5438 specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some default  
5439 value.

5440 If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key will  
5441 as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the derived  
5442 key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its **CKA\_SENSITIVE** attribute.

5443 Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
5444 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE,  
5445 then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite* value from its  
5446 **CKA\_EXTRACTABLE** attribute.

5447 For this mechanism, the ulMinKeySize and ulMaxKeySize fields of the **CK\_MECHANISM\_INFO** structure  
5448 both indicate 20 bytes.

5449 Note that this mechanism is only useable for key exchange suites that do not use a fixed length 20-byte  
5450 pre master secret key with an embedded version number. This includes the Diffie-Hellman and Elliptic  
5451 Curve Cryptography key exchange suites, but excludes the RSA key exchange suites.

## 5452 2.41.6 WTLS PRF (pseudorandom function)

5453 PRF (pseudo random function) in WTLS, denoted **CKM\_WTLS\_PRF**, is a mechanism used to produce a  
5454 securely generated pseudo-random output of arbitrary length. The keys it uses are generic secret keys.

5455 It has a parameter, a **CK\_WTLS\_PRF\_PARAMS** structure, which allows for passing the mechanism type  
5456 of the digest mechanism to be used, the passing of the input seed and its length, the passing of an  
5457 identifying label and its length and the passing of the length of the output to the token and for receiving  
5458 the output.

5459 This mechanism produces securely generated pseudo-random output of the length specified in the  
5460 parameter.

5461 This mechanism departs from the other key derivation mechanisms in Cryptoki in not using the template  
5462 sent along with this mechanism during a **C\_DeriveKey** function call, which means the template shall be a  
5463 NULL\_PTR. For most key-derivation mechanisms, **C\_DeriveKey** returns a single key handle as a result  
5464 of a successful completion. However, since the **CKM\_WTLS\_PRF** mechanism returns the requested  
5465 number of output bytes in the **CK\_WTLS\_PRF\_PARAMS** structure specified as the mechanism  
5466 parameter, the parameter *phKey* passed to **C\_DeriveKey** is unnecessary, and should be a NULL\_PTR.  
5467 If a call to **C\_DeriveKey** with this mechanism fails, then no output will be generated.

## 5468 2.41.7 Server Key and MAC derivation

5469 Server key, MAC and IV derivation in WTLS, denoted  
5470 **CKM\_WTLS\_SERVER\_KEY\_AND\_MAC\_DERIVE**, is a mechanism used to derive the appropriate  
5471 cryptographic keying material used by a cipher suite from the master secret key and random data. This  
5472 mechanism returns the key handles for the keys generated in the process, as well as the IV created.

5473 It has a parameter, a **CK\_WTLS\_KEY\_MAT\_PARAMS** structure, which allows for the passing of the  
5474 mechanism type of the digest mechanism to be used, random data, the characteristic of the cryptographic  
5475 material for the given cipher suite, and a pointer to a structure which receives the handles and IV which  
5476 were generated.

5477 This mechanism contributes to the creation of two distinct keys and returns one IV (if an IV is requested  
5478 by the caller) back to the caller. The keys are all given an object class of **CKO\_SECRET\_KEY**.

5479 The MACing key (server write MAC secret) is always given a type of **CKK\_GENERIC\_SECRET**. It is  
5480 flagged as valid for signing, verification and derivation operations.

5481 The other key (server write key) is typed according to information found in the template sent along with  
5482 this mechanism during a **C\_DeriveKey** function call. By default, it is flagged as valid for encryption,  
5483 decryption, and derivation operations.

5484 An IV (server write IV) will be generated and returned if the *ullVSizeInBits* field of the  
5485 **CK\_WTLS\_KEY\_MAT\_PARAMS** field has a nonzero value. If it is generated, its length in bits will agree  
5486 with the value in the *ullVSizeInBits* field

5487 Both keys inherit the values of the **CKA\_SENSITIVE**, **CKA\_ALWAYS\_SENSITIVE**,  
5488 **CKA\_EXTRACTABLE**, and **CKA\_NEVER\_EXTRACTABLE** attributes from the base key. The template  
5489 provided to **C\_DeriveKey** may not specify values for any of these attributes that differ from those held by  
5490 the base key.

5491 Note that the **CK\_WTLS\_KEY\_MAT\_OUT** structure pointed to by the **CK\_WTLS\_KEY\_MAT\_PARAMS**  
5492 structure's *pReturnedKeyMaterial* field will be modified by the **C\_DeriveKey** call. In particular, the two key  
5493 handle fields in the **CK\_WTLS\_KEY\_MAT\_OUT** structure will be modified to hold handles to the newly-  
5494 created keys; in addition, the buffer pointed to by the **CK\_WTLS\_KEY\_MAT\_OUT** structure's *pIV* field will  
5495 have the IV returned in them (if an IV is requested by the caller). Therefore, this field must point to a  
5496 buffer with sufficient space to hold any IV that will be returned.

5497 This mechanism departs from the other key derivation mechanisms in Cryptoki in its returned information.  
5498 For most key-derivation mechanisms, **C\_DeriveKey** returns a single key handle as a result of a  
5499 successful completion. However, since the **CKM\_WTLS\_SERVER\_KEY\_AND\_MAC\_DERIVE**  
5500 mechanism returns all of its key handles in the **CK\_WTLS\_KEY\_MAT\_OUT** structure pointed to by the  
5501 **CK\_WTLS\_KEY\_MAT\_PARAMS** structure specified as the mechanism parameter, the parameter *phKey*  
5502 passed to **C\_DeriveKey** is unnecessary, and should be a **NONE\_PTR**.

5503 If a call to **C\_DeriveKey** with this mechanism fails, then *none* of the two keys will be created.

## 5504 2.41.8 Client key and MAC derivation

5505 Client key, MAC and IV derivation in WTLS, denoted **CKM\_WTLS\_CLIENT\_KEY\_AND\_MAC\_DERIVE**,  
5506 is a mechanism used to derive the appropriate cryptographic keying material used by a cipher suite from  
5507 the master secret key and random data. This mechanism returns the key handles for the keys generated  
5508 in the process, as well as the IV created.

5509 It has a parameter, a **CK\_WTLS\_KEY\_MAT\_PARAMS** structure, which allows for the passing of the  
5510 mechanism type of the digest mechanism to be used, random data, the characteristic of the cryptographic  
5511 material for the given cipher suite, and a pointer to a structure which receives the handles and IV which  
5512 were generated.

5513 This mechanism contributes to the creation of two distinct keys and returns one IV (if an IV is requested  
5514 by the caller) back to the caller. The keys are all given an object class of **CKO\_SECRET\_KEY**.

5515 The MACing key (client write MAC secret) is always given a type of **CKK\_GENERIC\_SECRET**. It is  
5516 flagged as valid for signing, verification and derivation operations.

5517 The other key (client write key) is typed according to information found in the template sent along with this  
5518 mechanism during a **C\_DeriveKey** function call. By default, it is flagged as valid for encryption,  
5519 decryption, and derivation operations.

5520 An IV (client write IV) will be generated and returned if the *ullVSizeInBits* field of the  
5521 **CK\_WTLS\_KEY\_MAT\_PARAMS** field has a nonzero value. If it is generated, its length in bits will agree  
5522 with the value in the *ullVSizeInBits* field

5523 Both keys inherit the values of the **CKA\_SENSITIVE**, **CKA\_ALWAYS\_SENSITIVE**,  
5524 **CKA\_EXTRACTABLE**, and **CKA\_NEVER\_EXTRACTABLE** attributes from the base key. The template  
5525 provided to **C\_DeriveKey** may not specify values for any of these attributes that differ from those held by  
5526 the base key.

5527 Note that the **CK\_WTLS\_KEY\_MAT\_OUT** structure pointed to by the **CK\_WTLS\_KEY\_MAT\_PARAMS**  
5528 structure's *pReturnedKeyMaterial* field will be modified by the **C\_DeriveKey** call. In particular, the two key  
5529 handle fields in the **CK\_WTLS\_KEY\_MAT\_OUT** structure will be modified to hold handles to the newly-  
5530 created keys; in addition, the buffer pointed to by the **CK\_WTLS\_KEY\_MAT\_OUT** structure's *pIV* field will

5531 have the IV returned in them (if an IV is requested by the caller). Therefore, this field must point to a  
 5532 buffer with sufficient space to hold any IV that will be returned.  
 5533 This mechanism departs from the other key derivation mechanisms in Cryptoki in its returned information.  
 5534 For most key-derivation mechanisms, **C\_DeriveKey** returns a single key handle as a result of a  
 5535 successful completion. However, since the **CKM\_WTLS\_CLIENT\_KEY\_AND\_MAC\_DERIVE** mechanism  
 5536 returns all of its key handles in the **CK\_WTLS\_KEY\_MAT\_OUT** structure pointed to by the  
 5537 **CK\_WTLS\_KEY\_MAT\_PARAMS** structure specified as the mechanism parameter, the parameter *phKey*  
 5538 passed to **C\_DeriveKey** is unnecessary, and should be a **NONE\_PTR**.  
 5539 If a call to **C\_DeriveKey** with this mechanism fails, then *none* of the two keys will be created.

## 5540 2.42 SP 800-108 Key Derivation

5541 NIST SP800-108 defines three types of key derivation functions (KDF); a Counter Mode KDF, a  
 5542 Feedback Mode KDF and a Double Pipeline Mode KDF.  
 5543 This section defines a unique mechanism for each type of KDF. These mechanisms can be used to  
 5544 derive one or more symmetric keys from a single base symmetric key.  
 5545 The KDFs defined in SP800-108 are all built upon pseudo random functions (PRF). In general terms, the  
 5546 PRFs accept two pieces of input; a base key and some input data. The base key is taken from the  
 5547 *hBaseKey* parameter to **C\_Derive**. The input data is constructed from an iteration variable (internally  
 5548 defined by the KDF/PRF) and the data provided in the **CK\_PRF\_DATA\_PARAM** array that is part of the  
 5549 mechanism parameter.

5550 *Table 160, SP800-108 Mechanisms vs. Functions*

| Mechanism                         | Functions         |               |         |        |                    |               |        |
|-----------------------------------|-------------------|---------------|---------|--------|--------------------|---------------|--------|
|                                   | Encrypt & Decrypt | Sign & Verify | SR & VR | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SP800_108_COUNTER_KDF         |                   |               |         |        |                    |               | ✓      |
| CKM_SP800_108_FEEDBACK_KDF        |                   |               |         |        |                    |               | ✓      |
| CKM_SP800_108_DOUBLE_PIPELINE_KDF |                   |               |         |        |                    |               | ✓      |

5551  
 5552 For these mechanisms, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO**  
 5553 structure specify the minimum and maximum supported base key size in bits. Note, these mechanisms  
 5554 support multiple PRF types and key types; as such the values reported by *ulMinKeySize* and  
 5555 *ulMaxKeySize* specify the minimum and maximum supported base key size when all PRF and keys types  
 5556 are considered. For example, a Cryptoki implementation may support CKK\_GENERIC\_SECRET keys  
 5557 that can be as small as 8-bits in length and therefore *ulMinKeySize* could report 8-bits. However, for an  
 5558 AES-CMAC PRF the base key must be of type CKK\_AES and must be either 16-bytes, 24-bytes or 32-  
 5559 bytes in lengths and therefore the value reported by *ulMinKeySize* could be misleading. Depending on  
 5560 the PRF type selected, additional key size restrictions may apply.

### 5561 2.42.1 Definitions

5562 Mechanisms:

5563     CKM\_SP800\_108\_COUNTER\_KDF  
 5564     CKM\_SP800\_108\_FEEDBACK\_KDF  
 5565     CKM\_SP800\_108\_DOUBLE\_PIPELINE\_KDF

5566

5567 Data Field Types:

5568 CK\_SP800\_108\_ITERATION\_VARIABLE  
5569 CK\_SP800\_108\_COUNTER  
5570 CK\_SP800\_108\_DKM\_LENGTH  
5571 CK\_SP800\_108\_BYTE\_ARRAY  
5572  
5573 DKM Length Methods:  
5574 CK\_SP800\_108\_DKM\_LENGTH\_SUM\_OF\_KEYS  
5575 CK\_SP800\_108\_DKM\_LENGTH\_SUM\_OF\_SEGMENTS

## 5576 2.42.2 Mechanism Parameters

### 5577 ◆ CK\_SP800\_108\_PRF\_TYPE

5578 The **CK\_SP800\_108\_PRF\_TYPE** field of the mechanism parameter is used to specify the type of PRF  
5579 that is to be used. It is defined as follows:

```
5580 typedef CK_MECHANISM_TYPE CK_SP800_108_PRF_TYPE;
```

5581 The **CK\_SP800\_108\_PRF\_TYPE** field reuses the existing mechanisms definitions. The following table  
5582 lists the supported PRF types:

5583 *Table 161, SP800-108 Pseudo Random Functions*

| Pseudo Random Function Identifiers |
|------------------------------------|
| CKM_SHA_1_HMAC                     |
| CKM_SHA224_HMAC                    |
| CKM_SHA256_HMAC                    |
| CKM_SHA384_HMAC                    |
| CKM_SHA512_HMAC                    |
| CKM_SHA3_224_HMAC                  |
| CKM_SHA3_256_HMAC                  |
| CKM_SHA3_384_HMAC                  |
| CKM_SHA3_512_HMAC                  |
| CKM_DES3_CMAC                      |
| CKM_AES_CMAC                       |

5584

### 5585 ◆ CK\_PRF\_DATA\_TYPE

5586 Each mechanism parameter contains an array of **CK\_PRF\_DATA\_PARAM** structures. The  
5587 **CK\_PRF\_DATA\_PARAM** structure contains **CK\_PRF\_DATA\_TYPE** field. The **CK\_PRF\_DATA\_TYPE**  
5588 field is used to identify the type of data identified by each **CK\_PRF\_DATA\_PARAM** element in the array.  
5589 Depending on the type of KDF used, some data field types are mandatory, some data field types are  
5590 optional and some data field types are not allowed. These requirements are defined on a per-mechanism  
5591 basis in the sections below. The **CK\_PRF\_DATA\_TYPE** is defined as follows:

```
5592 typedef CK ULONG CK_PRF_DATA_TYPE;
```

5593 The following table lists all of the supported data field types:

5594 Table 162, SP800-108 PRF Data Field Types

| Data Field Identifier           | Description                                                                                                                                                                                                                                 |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CK_SP800_108_ITERATION_VARIABLE | Identifies the iteration variable defined internally by the KDF.                                                                                                                                                                            |
| CK_SP800_108_COUNTER            | Identifies an optional counter value represented as a binary string. Exact formatting of the counter value is defined by the CK_SP800_108_COUNTER_FORMAT structure. The value of the counter is defined by the KDF's internal loop counter. |
| CK_SP800_108_DKM_LENGTH         | Identifies the length in bits of the derived keying material (DKM) represented as a binary string. Exact formatting of the length value is defined by the CK_SP800_108_DKM_LENGTH_FORMAT structure.                                         |
| CK_SP800_108_BYTE_ARRAY         | Identifies a generic byte array of data. This data type can be used to provide "context", "label", "separator bytes" as well as any other type of encoding information required by the higher level protocol.                               |

5595

## 5596 ◆ CK\_PRF\_DATA\_PARAM

5597 **CK\_PRF\_DATA\_PARAM** is used to define a segment of input for the PRF. Each mechanism parameter  
 5598 supports an array of **CK\_PRF\_DATA\_PARAM** structures. The **CK\_PRF\_DATA\_PARAM** is defined as  
 5599 follows:

```
5600 typedef struct CK_PRF_DATA_PARAM
5601 {
5602 CK_PRF_DATA_TYPE type;
5603 CK_VOID_PTR pValue;
5604 CK ULONG ulValueLen;
5605 } CK_PRF_DATA_PARAM;
5606
5607 typedef CK_PRF_DATA_PARAM CK_PTR CK_PRF_DATA_PARAM_PTR
5608
```

5609 The fields of the **CK\_PRF\_DATA\_PARAM** structure have the following meaning:

5610           **type**      *defines the type of data pointed to by pValue*

5611           **pValue**     *pointer to the data defined by type*

5612           **ulValueLen**   *size of the data pointed to by pValue*

5613 If the **type** field of the **CK\_PRF\_DATA\_PARAM** structure is set to  
 5614 CK\_SP800\_108\_ITERATION\_VARIABLE, then **pValue** must be set the appropriate value for the KDF's  
 5615 iteration variable type. For the Counter Mode KDF, **pValue** must be assigned a valid  
 5616 CK\_SP800\_108\_COUNTER\_FORMAT\_PTR and **ulValueLen** must be set to  
 5617 sizeof(CK\_SP800\_108\_COUNTER\_FORMAT). For all other KDF types, **pValue** must be set to  
 5618 NULL\_PTR and **ulValueLen** must be set to 0.

5619

5620 If the **type** field of the **CK\_PRF\_DATA\_PARAM** structure is set to CK\_SP800\_108\_COUNTER, then  
 5621 **pValue** must be assigned a valid CK\_SP800\_108\_COUNTER\_FORMAT\_PTR and **ulValueLen** must be  
 5622 set to sizeof(CK\_SP800\_108\_COUNTER\_FORMAT).

5623

5624 If the *type* field of the **CK\_PRF\_DATA\_PARAM** structure is set to CK\_SP800\_108\_DKM\_LENGTH then  
 5625 *pValue* must be assigned a valid CK\_SP800\_108\_DKM\_LENGTH\_FORMAT\_PTR and *ulValueLen* must  
 5626 be set to sizeof(CK\_SP800\_108\_DKM\_LENGTH\_FORMAT).

5627

5628 If the *type* field of the **CK\_PRF\_DATA\_PARAM** structure is set to CK\_SP800\_108\_BYTE\_ARRAY, then  
 5629 *pValue* must be assigned a valid CK\_BYTE\_PTR value and *ulValueLen* must be set to a non-zero length.

5630 ◆ **CK\_SP800\_108\_COUNTER\_FORMAT**

5631 **CK\_SP800\_108\_COUNTER\_FORMAT** is used to define the encoding format for a counter value. The  
 5632 **CK\_SP800\_108\_COUNTER\_FORMAT** is defined as follows:

```

5633 typedef struct CK_SP800_108_COUNTER_FORMAT
5634 {
5635 CK_BBOOL bLittleEndian;
5636 CK ULONG ulWidthInBits;
5637 } CK_SP800_108_COUNTER_FORMAT;
5638
5639 typedef CK_SP800_108_COUNTER_FORMAT CK_PTR
5640 CK_SP800_108_COUNTER_FORMAT_PTR

```

5641

5642 The fields of the CK\_SP800\_108\_COUNTER\_FORMAT structure have the following meaning:  
 5643       **bLittleEndian**     *defines if the counter should be represented in Big Endian or Little*  
 5644       *Endian format*

5645       **ulWidthInBits**    *defines the number of bits used to represent the counter value*

5646 ◆ **CK\_SP800\_108\_DKM\_LENGTH\_METHOD**

5647 **CK\_SP800\_108\_DKM\_LENGTH\_METHOD** is used to define how the DKM length value is calculated.  
 5648 The **CK\_SP800\_108\_DKM\_LENGTH\_METHOD** type is defined as follows:

```

5649 typedef CK ULONG CK_SP800_108_DKM_LENGTH_METHOD;

```

5650 The following table lists all of the supported DKM Length Methods:

5651 *Table 163, SP800-108 DKM Length Methods*

| <b>DKM Length Method Identifier</b>     | <b>Description</b>                                                                                                                                |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| CK_SP800_108_DKM_LENGTH_SUM_OF_KEYS     | Specifies that the DKM length should be set to the sum of the length of all keys derived by this invocation of the KDF.                           |
| CK_SP800_108_DKM_LENGTH_SUM_OF_SEGMENTS | Specifies that the DKM length should be set to the sum of the length of all segments of output produced by the PRF by this invocation of the KDF. |

5652

5653 ◆ **CK\_SP800\_108\_DKM\_LENGTH\_FORMAT**

5654 **CK\_SP800\_108\_DKM\_LENGTH\_FORMAT** is used to define the encoding format for the DKM length  
 5655 value. The **CK\_SP800\_108\_DKM\_LENGTH\_FORMAT** is defined as follows:

```

5656 typedef struct CK_SP800_108_DKM_LENGTH_FORMAT

```

```

5657 {
5658 CK_SP800_108_DKM_LENGTH_METHOD dkmLengthMethod;
5659 CK_BBOOL bLittleEndian;
5660 CK ULONG ulWidthInBits;
5661 } CK_SP800_108_DKM_LENGTH_FORMAT;
5662
5663 typedef CK_SP800_108_DKM_LENGTH_FORMAT CK_PTR
5664 CK_SP800_108_DKM_LENGTH_FORMAT_PTR
5665
5666 The fields of the CK_SP800_108_DKM_LENGTH_FORMAT structure have the following meaning:
5667 dkmLengthMethod defines the method used to calculate the DKM length value
5668
5669 bLittleEndian defines if the DKM length value should be represented in Big
5670 Endian or LittleEndian format
5671 ulWidthInBits defines the number of bits used to represent the DKM length value

```

## 5671 ◆ CK\_DERIVED\_KEY

5672 **CK\_DERIVED\_KEY** is used to define an additional key to be derived as well as provide a  
 5673 **CK\_OBJECT\_HANDLE\_PTR** to receive the handle for the derived keys. The **CK\_DERIVED\_KEY** is  
 5674 defined as follows:

```

5675 typedef struct CK_DERIVED_KEY
5676 {
5677 CK_ATTRIBUTE_PTR pTemplate;
5678 CK ULONG ulAttributeCount;
5679 CK_OBJECT_HANDLE_PTR phKey;
5680 } CK_DERIVED_KEY;
5681
5682 typedef CK_DERIVED_KEY CK_PTR CK_DERIVED_KEY_PTR
5683

```

5684 The fields of the **CK\_DERIVED\_KEY** structure have the following meaning:  
 5685 pTemplate pointer to a template that defines a key to derive  
 5686 ulAttributeCount number of attributes in the template pointed to by pTemplate  
 5687 phKey pointer to receive the handle for a derived key

## 5688 ◆ CK\_SP800\_108\_KDF\_PARAMS, CK\_SP800\_108\_KDF\_PARAMS\_PTR

5689 **CK\_SP800\_108\_KDF\_PARAMS** is a structure that provides the parameters for the  
 5690 **CKM\_SP800\_108\_COUNTER\_KDF** and **CKM\_SP800\_108\_DOUBLE\_PIPELINE\_KDF** mechanisms.

```

5691
5692 typedef struct CK_SP800_108_KDF_PARAMS
5693 {
5694 CK_SP800_108_PRF_TYPE prfType;
5695 CK ULONG ulNumberOfDataParams;
5696 CK_PRF_DATA_PARAM_PTR pDataParams;
5697 CK ULONG ulAdditionalDerivedKeys;

```

```

5698 CK_DERIVED_KEY_PTR pAdditionalDerivedKeys;
5699 } CK_SP800_108_KDF_PARAMS;
5700
5701 typedef CK_SP800_108_KDF_PARAMS CK_PTR
5702 CK_SP800_108_KDF_PARAMS_PTR;
5703
5704 The fields of the CK_SP800_108_KDF_PARAMS structure have the following meaning:
5705 prfType type of PRF
5706 ulNumberOfDataParams number of elements in the array pointed to by pDataParams
5707 pDataParams an array of CK_PRF_DATA_PARAM structures. The array defines
5708 input parameters that are used to construct the "data" input to the
5709 PRF.
5710 ulAdditionalDerivedKeys number of additional keys that will be derived and the number of
5711 elements in the array pointed to by pAdditionalDerivedKeys. If
5712 pAdditionalDerivedKeys is set to NULL_PTR, this parameter must
5713 be set to 0.
5714 pAdditionalDerivedKeys an array of CK_DERIVED_KEY structures. If
5715 ulAdditionalDerivedKeys is set to 0, this parameter must be set to
5716 NULL_PTR

```

◆ **CK\_SP800\_108\_FEEDBACK\_KDF\_PARAMS,**  
**CK\_SP800\_108\_FEEDBACK\_KDF\_PARAMS\_PTR**

5719 The **CK\_SP800\_108\_FEEDBACK\_KDF\_PARAMS** structure provides the parameters for the
5720 CKM\_SP800\_108\_FEEDBACK\_KDF mechanism. It is defined as follows:

```

5721 typedef struct CK_SP800_108_FEEDBACK_KDF_PARAMS
5722 {
5723 CK_SP800_108_PRF_TYPE prfType;
5724 CK ULONG ulNumberOfDataParams;
5725 CK_PRF_DATA_PARAM_PTR pDataParams;
5726 CK ULONG ulIVLen;
5727 CK_BYTE_PTR pIV;
5728 CK ULONG ulAdditionalDerivedKeys;
5729 CK_DERIVED_KEY_PTR pAdditionalDerivedKeys;
5730 } CK_SP800_108_FEEDBACK_KDF_PARAMS;
5731
5732 typedef CK_SP800_108_FEEDBACK_KDF_PARAMS CK_PTR
5733 CK_SP800_108_FEEDBACK_KDF_PARAMS_PTR;
5734

```

5735 The fields of the **CK\_SP800\_108\_FEEDBACK\_KDF\_PARAMS** structure have the following meaning:
5736 prfType type of PRF

```

5737 ulNumberOfDataParams number of elements in the array pointed to by pDataParams
5738 pDataParams an array of CK_PRF_DATA_PARAM structures. The array defines
5739 input parameters that are used to construct the "data" input to the
5740 PRF.

```

|      |                                |                                                                                                                                                                                                                    |
|------|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5741 | <i>ullVLen</i>                 | <i>the length in bytes of the IV. If pIV is set to NULL_PTR, this parameter must be set to 0.</i>                                                                                                                  |
| 5742 |                                |                                                                                                                                                                                                                    |
| 5743 | <i>pIV</i>                     | <i>an array of bytes to be used as the IV for the feedback mode KDF. This parameter is optional and can be set to NULL_PTR. If ullVLen is set to 0, this parameter must be set to NULL_PTR.</i>                    |
| 5744 |                                |                                                                                                                                                                                                                    |
| 5745 |                                |                                                                                                                                                                                                                    |
| 5746 | <i>ulAdditionalDerivedKeys</i> | <i>number of additional keys that will be derived and the number of elements in the array pointed to by pAdditionalDerivedKeys. If pAdditionalDerivedKeys is set to NULL_PTR, this parameter must be set to 0.</i> |
| 5747 |                                |                                                                                                                                                                                                                    |
| 5748 |                                |                                                                                                                                                                                                                    |
| 5749 |                                |                                                                                                                                                                                                                    |
| 5750 | <i>pAdditionalDerivedKeys</i>  | <i>an array of CK_DERIVED_KEY structures. If ulAdditionalDerivedKeys is set to 0, this parameter must be set to NULL_PTR.</i>                                                                                      |
| 5751 |                                |                                                                                                                                                                                                                    |
| 5752 |                                |                                                                                                                                                                                                                    |

### 5753 2.42.3 Counter Mode KDF

5754 The SP800-108 Counter Mode KDF mechanism, denoted **CKM\_SP800\_108\_COUNTER\_KDF**,  
 5755 represents the KDF defined SP800-108 section 5.1. **CKM\_SP800\_108\_COUNTER\_KDF** is a  
 5756 mechanism for deriving one or more symmetric keys from a symmetric base key.

5757 It has a parameter, a **CK\_SP800\_108\_KDF\_PARAMS** structure.

5758 The following table lists the data field types that are supported for this KDF type and their meaning:

5759 *Table 164, Counter Mode data field requirements*

| Data Field Identifier           | Description                                                                                                                                                                                                                                                                                                 |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CK_SP800_108_ITERATION_VARIABLE | This data field type is mandatory.<br>This data field type identifies the location of the iteration variable in the constructed PRF input data.<br>The iteration variable for this KDF type is a counter.<br>Exact formatting of the counter value is defined by the CK_SP800_108_COUNTER_FORMAT structure. |
| CK_SP800_108_COUNTER            | This data field type is invalid for this KDF type.                                                                                                                                                                                                                                                          |
| CK_SP800_108_DKM_LENGTH         | This data field type is optional.<br>This data field type identifies the location of the DKM length in the constructed PRF input data.<br>Exact formatting of the DKM length is defined by the CK_SP800_108_DKM_LENGTH_FORMAT structure.<br>If specified, only one instance of this type may be specified.  |
| CK_SP800_108_BYTE_ARRAY         | This data field type is optional.<br>This data field type identifies the location and value of a byte array of data in the constructed PRF input data.<br>This standard does not restrict the number of instances of this data type.                                                                        |

5760

5761 SP800-108 limits the amount of derived keying material that can be produced by a Counter Mode KDF by  
 5762 limiting the internal loop counter to  $(2^r - 1)$ , where "r" is the number of bits used to represent the counter.  
 5763 Therefore the maximum number of bits that can be produced is  $(2^r - 1)h$ , where "h" is the length in bits of  
 5764 the output of the selected PRF.

## 2.42.4 Feedback Mode KDF

The SP800-108 Feedback Mode KDF mechanism, denoted **CKM\_SP800\_108\_FEEDBACK\_KDF**, represents the KDF defined SP800-108 section 5.2. **CKM\_SP800\_108\_FEEDBACK\_KDF** is a mechanism for deriving one or more symmetric keys from a symmetric base key.

It has a parameter, a **CK\_SP800\_108\_FEEDBACK\_KDF\_PARAMS** structure.

The following table lists the data field types that are supported for this KDF type and their meaning:

*Table 165, Feedback Mode data field requirements*

| Data Field Identifier           | Description                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CK_SP800_108_ITERATION_VARIABLE | This data field type is mandatory.<br>This data field type identifies the location of the iteration variable in the constructed PRF input data.<br>The iteration variable is defined as K(i-1) in section 5.2 of SP800-108.<br>The size, format and value of this data input is defined by the internal KDF structure and PRF output.<br>Exact formatting of the counter value is defined by the CK_SP800_108_COUNTER_FORMAT structure. |
| CK_SP800_108_COUNTER            | This data field type is optional.<br>This data field type identifies the location of the counter in the constructed PRF input data.<br>Exact formatting of the counter value is defined by the CK_SP800_108_COUNTER_FORMAT structure.<br>If specified, only one instance of this type may be specified.                                                                                                                                 |
| CK_SP800_108_DKM_LENGTH         | This data field type is optional.<br>This data field type identifies the location of the DKM length in the constructed PRF input data.<br>Exact formatting of the DKM length is defined by the CK_SP800_108_DKM_LENGTH_FORMAT structure.<br>If specified, only one instance of this type may be specified.                                                                                                                              |
| CK_SP800_108_BYTE_ARRAY         | This data field type is optional.<br>This data field type identifies the location and value of a byte array of data in the constructed PRF input data.<br>This standard does not restrict the number of instances of this data type.                                                                                                                                                                                                    |

5772

SP800-108 limits the amount of derived keying material that can be produced by a Feedback Mode KDF by limiting the internal loop counter to  $(2^{32}-1)$ . Therefore the maximum number of bits that can be produced is  $(2^{32}-1)h$ , where "h" is the length in bits of the output of the selected PRF.

## 2.42.5 Double Pipeline Mode KDF

The SP800-108 Double Pipeline Mode KDF mechanism, denoted **CKM\_SP800\_108\_DOUBLE\_PIPELINE\_KDF**, represents the KDF defined SP800-108 section 5.3. **CKM\_SP800\_108\_DOUBLE\_PIPELINE\_KDF** is a mechanism for deriving one or more symmetric keys from a symmetric base key.

It has a parameter, a **CK\_SP800\_108\_KDF\_PARAMS** structure.

The following table lists the data field types that are supported for this KDF type and their meaning:

5783 Table 166, Double Pipeline Mode data field requirements

| Data Field Identifier           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CK_SP800_108_ITERATION_VARIABLE | <p>This data field type is mandatory.</p> <p>This data field type identifies the location of the iteration variable in the constructed PRF input data.</p> <p>The iteration variable is defined as A(i) in section 5.3 of SP800-108.</p> <p>The size, format and value of this data input is defined by the internal KDF structure and PRF output.</p> <p>Exact formatting of the counter value is defined by the CK_SP800_108_COUNTER_FORMAT structure.</p> |
| CK_SP800_108_COUNTER            | <p>This data field type is optional.</p> <p>This data field type identifies the location of the counter in the constructed PRF input data.</p> <p>Exact formatting of the counter value is defined by the CK_SP800_108_COUNTER_FORMAT structure.</p> <p>If specified, only one instance of this type may be specified.</p>                                                                                                                                   |
| CK_SP800_108_DKM_LENGTH         | <p>This data field type is optional.</p> <p>This data field type identifies the location of the DKM length in the constructed PRF input data.</p> <p>Exact formatting of the DKM length is defined by the CK_SP800_108_DKM_LENGTH_FORMAT structure.</p> <p>If specified, only one instance of this type may be specified.</p>                                                                                                                                |
| CK_SP800_108_BYTE_ARRAY         | <p>This data field type is optional.</p> <p>This data field type identifies the location and value of a byte array of data in the constructed PRF input data.</p> <p>This standard does not restrict the number of instances of this data type.</p>                                                                                                                                                                                                          |

5784

5785 SP800-108 limits the amount of derived keying material that can be produced by a Double-Pipeline Mode  
 5786 KDF by limiting the internal loop counter to  $(2^{32}-1)$ . Therefore the maximum number of bits that can be  
 5787 produced is  $(2^{32}-1)h$ , where "h" is the length in bits of the output of the selected PRF.

5788 The Double Pipeline KDF requires an internal IV value. The IV is constructed using the same method  
 5789 used to construct the PRF input data; the data/values identified by the array of CK\_PRF\_DATA\_PARAM  
 5790 structures are concatenated in to a byte array that is used as the IV. As shown in SP800-108 section 5.3,  
 5791 the CK\_SP800\_108\_ITERATION\_VARIABLE and CK\_SP800\_108\_COUNTER data field types are not  
 5792 included in IV construction process. All other data field types are included in the construction process.

## 5793 2.42.6 Deriving Additional Keys

5794 The KDFs defined in this section can be used to derive more than one symmetric key from the base key.  
 5795 The **C\_Derive** function accepts one CK\_ATTRIBUTE\_PTR to define a single derived key and one  
 5796 CK\_OBJECT\_HANDLE\_PTR to receive the handle for the derived key.

5797 To derive additional keys, the mechanism parameter structure can be filled in with one or more  
 5798 CK\_DERIVED\_KEY structures. Each structure contains a CK\_ATTRIBUTE\_PTR to define a derived key  
 5799 and a CK\_OBJECT\_HANDLE\_PTR to receive the handle for the additional derived keys. The key  
 5800 defined by the **C\_Derive** function parameters is always derived before the keys defined by the  
 5801 CK\_DERIVED\_KEY array that is part of the mechanism parameter. The additional keys that are defined  
 5802 by the CK\_DERIVED\_KEY array are derived in the order they are defined in the array. That is to say that  
 5803 the derived keying material produced by the KDF is processed from left to right, and bytes are assigned

5804 first to the key defined by the **C\_Derive** function parameters, and then bytes are assigned to the keys that  
 5805 are defined by the CK\_DERIVED\_KEY array in the order they are defined in the array.  
 5806 Each internal iteration of a KDF produces a unique segment of PRF output. Sometimes, a single iteration  
 5807 will produce enough keying material for the key being derived. Other times, additional internal iterations  
 5808 are performed to produce multiple segments which are concatenated together to produce enough keying  
 5809 material for the derived key(s).  
 5810 When deriving multiple keys, no key can be created using part of a segment that was used for another  
 5811 key. All keys must be created from disjoint segments. For example, if the parameters are defined such  
 5812 that a 48-byte key (defined by the **C\_Derive** function parameters) and a 16-byte key (defined by the  
 5813 content of CK\_DERIVED\_KEY) are to be derived using **CKM\_SHA256\_HMAC** as a PRF, three internal  
 5814 iterations of the KDF will be performed and three segments of PRF output will be produced. The first  
 5815 segment and half of the second segment will be used to create the 48-byte key and the third segment will  
 5816 be used to create the 16-byte key.

|                           |                 |                 |                 |
|---------------------------|-----------------|-----------------|-----------------|
| 3 KDF Segments of Output: | 32-byte segment | 32-byte segment | 32-byte segment |
|---------------------------|-----------------|-----------------|-----------------|

|                 |             |        |             |        |
|-----------------|-------------|--------|-------------|--------|
| 2 Derived Keys: | 48-byte key | unused | 16-byte key | unused |
|-----------------|-------------|--------|-------------|--------|

5817 In the above example, if the CK\_SP800\_108\_DKM\_LENGTH data field type is specified with method  
 5818 CK\_SP800\_108\_DKM\_LENGTH\_SUM\_OF\_KEYS, then the DKM length value will be 512 bits. If the  
 5819 CK\_SP800\_108\_DKM\_LENGTH data field type is specified with method  
 5820 CK\_SP800\_108\_DKM\_LENGTH\_SUM\_OF\_SEGMENTS, then the DKM length value will be 768 bits.  
 5821 When deriving multiple keys, if any of the keys cannot be derived for any reason, none of the keys shall  
 5822 be derived. If the failure was caused by the content of a specific key's template (ie the template defined  
 5823 by the content of *pTemplate*), the corresponding *phKey* value will be set to CK\_INVALID\_HANDLE to  
 5824 identify the offending template.

## 5827 2.42.7 Key Derivation Attribute Rules

5828 The **CKM\_SP800\_108\_COUNTER\_KDF**, **CKM\_SP800\_108\_FEEDBACK\_KDF** and  
 5829 **CKM\_SP800\_108\_DOUBLE\_PIPELINE\_KDF** mechanisms have the following rules about key sensitivity  
 5830 and extractability:

- 5831 • The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key(s) can  
 5832 both be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on  
 5833 some default value.
- 5834 • If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
 5835 will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the  
 5836 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
 5837 **CKA\_SENSITIVE** attribute.
- 5838 • Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
 5839 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
 5840 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
 5841 value from its **CKA\_EXTRACTABLE** attribute.

## 5842 2.42.8 Constructing PRF Input Data

5843 SP800-108 defines the PRF input data for each KDF at a high level using terms like "label", "context",  
 5844 "separator", "counter"...etc. The value, formatting and order of the input data is not strictly defined by  
 5845 SP800-108, instead it is described as being defined by the "encoding scheme".

5846 To support any encoding scheme, these mechanisms construct the PRF input data from the array of  
 5847 CK\_PRF\_DATA\_PARAM structures in the mechanism parameter. All of the values defined by the  
 5848 CK\_PRF\_DATA\_PARAM array are concatenated in the order they are defined and passed in to the PRF  
 5849 as the data parameter.

### 5850 2.42.8.1 Sample Counter Mode KDF

5851 SP800-108 section 5.1 outlines a sample Counter Mode KDF which defines the following PRF input:

5852  $\text{PRF}(\text{KI}, [i]_2 \parallel \text{Label} \parallel 0x00 \parallel \text{Context} \parallel [L]_2)$

5853 Section 5.1 does not define the number of bits used to represent the counter (the “r” value) or the DKM  
5854 length (the “L” value), so 16-bits is assumed for both cases. The following sample code shows how to  
5855 define this PRF input data using an array of CK\_PRF\_DATA\_PARAM structures.

```
5856 #define DIM(a) (sizeof((a))/sizeof((a)[0]))
5857
5858 CK_OBJECT_HANDLE hBaseKey;
5859 CK_OBJECT_HANDLE hDerivedKey;
5860 CK_ATTRIBUTE derivedKeyTemplate = { ... };
5861
5862 CK_BYTE baLabel[] = {0xde, 0xad, 0xbe, 0xef};
5863 CK ULONG ulLabelLen = sizeof(baLabel);
5864 CK_BYTE baContext[] = {0xfe, 0xed, 0xbe, 0xef};
5865 CK ULONG ulContextLen = sizeof(baContext);
5866
5867 CK_SP800_108_COUNTER_FORMAT counterFormat = {0, 16};
5868 CK_SP800_108_DKM_LENGTH_FORMAT dkmFormat
5869 = {CK_SP800_108_DKM_LENGTH_SUM_OF_KEYS, 0, 16};
5870
5871 CK_PRF_DATA_PARAM dataParams[] =
5872 {
5873 { CK_SP800_ITERATION_VARIABLE,
5874 &counterFormat, sizeof(counterFormat) },
5875 { CK_SP800_108_BYTE_ARRAY, baLabel, ulLabelLen },
5876 { CK_SP800_108_BYTE_ARRAY, {0x00}, 1 },
5877 { CK_SP800_108_BYTE_ARRAY, baContext, ulContextLen },
5878 { CK_SP800_DKM_LENGTH, dkmFormat, sizeof(dkmFormat) }
5879 };
5880
5881 CK_SP800_108_KDF_PARAMS kdfParams =
5882 {
5883 CKM_AES_CMAC,
5884 DIM(dataParams),
5885 &dataParams,
5886 0, /* no addition derived keys */
5887 NULL /* no addition derived keys */
5888 };
5889
5890 CK_MECHANISM = mechanism
5891 {
5892 CKM_SP800_108_COUNTER_KDF,
5893 &kdfParams,
5894 sizeof(kdfParams)
5895 };
5896
5897 hBaseKey = GetBaseKeyHandle(.);
5898
5899 rv = C_DeriveKey(
5900 hSession,
5901 &mechanism,
5902 hBaseKey,
5903 &derivedKeyTemplate,
5904 DIM(derivedKeyTemplate),
5905 &hDerivedKey);
```

## 2.42.8.2 Sample SCP03 Counter Mode KDF

The SCP03 standard defines a variation of a counter mode KDF which defines the following PRF input:

$\text{PRF}(\text{Ki}, \text{Label} \parallel 0x00 \parallel [\text{L}]_2 \parallel [\text{i}]_2 \parallel \text{Context})$

SCP03 defines the number of bits used to represent the counter (the “r” value) and number of bits used to represent the DKM length (the “L” value) as 16-bits. The following sample code shows how to define this PRF input data using an array of CK\_PRF\_DATA\_PARAM structures.

```
5906
5907
5908
5909
5910
5911
5912 #define DIM(a) (sizeof((a))/sizeof((a)[0]))
5913
5914 CK_OBJECT_HANDLE hBaseKey;
5915 CK_OBJECT_HANDLE hDerivedKey;
5916 CK_ATTRIBUTE derivedKeyTemplate = { ... };
5917
5918 CK_BYTE baLabel[] = {0xde, 0xad, 0xbe , 0xef};
5919 CK ULONG ulLabelLen = sizeof(baLabel);
5920 CK_BYTE baContext[] = {0xfe, 0xed, 0xbe , 0xef};
5921 CK ULONG ulContextLen = sizeof(baContext);
5922
5923 CK_SP800_108_COUNTER_FORMAT counterFormat = {0, 16};
5924 CK_SP800_108_DKM_LENGTH_FORMAT dkmFormat
5925 = {CK_SP800_108_DKM_LENGTH_SUM_OF_KEYS, 0, 16};
5926
5927 CK_PRF_DATA_PARAM dataParams[] =
5928 {
5929 { CK_SP800_108_BYTE_ARRAY, baLabel, ulLabelLen },
5930 { CK_SP800_108_BYTE_ARRAY, {0x00}, 1 },
5931 { CK_SP800_108_DKM_LENGTH, dkmFormat, sizeof(dkmFormat) },
5932 { CK_SP800_108_ITERATION_VARIABLE,
5933 &counterFormat, sizeof(counterFormat) },
5934 { CK_SP800_108_BYTE_ARRAY, baContext, ulContextLen }
5935 };
5936
5937 CK_SP800_108_KDF_PARAMS kdfParams =
5938 {
5939 CKM_AES_CMAC,
5940 DIM(dataParams),
5941 &dataParams,
5942 0, /* no addition derived keys */
5943 NULL /* no addition derived keys */
5944 };
5945
5946 CK_MECHANISM = mechanism
5947 {
5948 CKM_SP800_108_COUNTER_KDF,
5949 &kdfParams,
5950 sizeof(kdfParams)
5951 };
5952
5953 hBaseKey = GetBaseKeyHandle(.....);
5954
5955 rv = C_DeriveKey(
5956 hSession,
5957 &mechanism,
5958 hBaseKey,
5959 &derivedKeyTemplate,
5960 DIM(derivedKeyTemplate),
5961 &hDerivedKey);
```

### 2.42.8.3 Sample Feedback Mode KDF

SP800-108 section 5.2 outlines a sample Feedback Mode KDF which defines the following PRF input:

$$\text{PRF}(K_i, K(i-1) \{\mid [i]_2\} \mid Label \mid 0x00 \mid Context \mid [L]_2)$$

Section 5.2 does not define the number of bits used to represent the counter (the “r” value) or the DKM length (the “L” value), so 16-bits is assumed for both cases. The counter is defined as being optional and is included in this example. The following sample code shows how to define this PRF input data using an array of CK\_PRF\_DATA\_PARAM structures.

```
5962 #define DIM(a) (sizeof((a))/sizeof((a)[0]))
5963
5964 CK_OBJECT_HANDLE hBaseKey;
5965 CK_OBJECT_HANDLE hDerivedKey;
5966 CK_ATTRIBUTE derivedKeyTemplate = { ... };
5967
5968 CK_BYTE baFeedbackIV[] = {0x01, 0x02, 0x03, 0x04};
5969 CK ULONG ulFeedbackIVLen = sizeof(baFeedbackIV);
5970 CK_BYTE baLabel[] = {0xde, 0xad, 0xbe, 0xef};
5971 CK ULONG ulLabelLen = sizeof(baLabel);
5972 CK_BYTE baContext[] = {0xfe, 0xed, 0xbe, 0xef};
5973 CK ULONG ulContextLen = sizeof(baContext);
5974
5975 CK_SP800_108_COUNTER_FORMAT counterFormat = {0, 16};
5976 CK_SP800_108_DKM_LENGTH_FORMAT dkmFormat
5977 = {CK_SP800_108_DKM_LENGTH_SUM_OF_KEYS, 0, 16};
5978
5979 CK_PRF_DATA_PARAM dataParams[] =
5980 {
5981 { CK_SP800_108_ITERATION_VARIABLE,
5982 &counterFormat, sizeof(counterFormat) },
5983 { CK_SP800_108_BYTE_ARRAY, baLabel, ulLabelLen },
5984 { CK_SP800_108_BYTE_ARRAY, {0x00}, 1 },
5985 { CK_SP800_108_BYTE_ARRAY, baContext, ulContextLen },
5986 { CK_SP800_108_DKM_LENGTH, dkmFormat, sizeof(dkmFormat) }
5987 };
5988
5989 CK_SP800_108_FEEDBACK_KDF_PARAMS kdfParams =
5990 {
5991 CKM_AES_CMAC,
5992 DIM(dataParams),
5993 &dataParams,
5994 ulFeedbackIVLen,
5995 baFeedbackIV,
5996 0, /* no addition derived keys */
5997 NULL /* no addition derived keys */
5998 };
5999
6000 CK_MECHANISM = mechanism
6001 {
6002 CKM_SP800_108_FEEDBACK_KDF,
6003 &kdfParams,
6004 sizeof(kdfParams)
6005 };
6006
6007 hBaseKey = GetBaseKeyHandle(.);
6008
6009 rv = C_DeriveKey(
```

```

6017 hSession,
6018 &mechanism,
6019 hBaseKey,
6020 &derivedKeyTemplate,
6021 DIM(derivedKeyTemplate),
6022 &hDerivedKey);

```

#### 6023 2.42.8.4 Sample Double-Pipeline Mode KDF

6024 SP800-108 section 5.3 outlines a sample Double-Pipeline Mode KDF which defines the two following  
 6025 PRF inputs:

```

6026 PRF (KI, A(i-1))
6027 PRF (KI, K(i-1) {|| [i]2 }|| Label || 0x00 || Context || [L]2)

```

6028 Section 5.3 does not define the number of bits used to represent the counter (the “r” value) or the DKM  
 6029 length (the “L” value), so 16-bits is assumed for both cases. The counter is defined as being optional so it  
 6030 is left out in this example. The following sample code shows how to define this PRF input data using an  
 6031 array of CK\_PRF\_DATA\_PARAM structures.

```

6032 #define DIM(a) (sizeof((a))/sizeof((a)[0]))
6033
6034 CK_OBJECT_HANDLE hBaseKey;
6035 CK_OBJECT_HANDLE hDerivedKey;
6036 CK_ATTRIBUTE derivedKeyTemplate = { ... };
6037
6038 CK_BYTE baLabel[] = {0xde, 0xad, 0xbe, 0xef};
6039 CK ULONG ulLabelLen = sizeof(baLabel);
6040 CK_BYTE baContext[] = {0xfe, 0xed, 0xbe, 0xef};
6041 CK ULONG ulContextLen = sizeof(baContext);
6042
6043 CK_SP800_108_DKM_LENGTH_FORMAT dkmFormat
6044 = {CK_SP800_108_DKM_LENGTH_SUM_OF_KEYS, 0, 16};
6045
6046 CK_PRF_DATA_PARAM dataParams[] =
6047 {
6048 { CK_SP800_108_BYTE_ARRAY, baLabel, ulLabelLen },
6049 { CK_SP800_108_BYTE_ARRAY, {0x00}, 1 },
6050 { CK_SP800_108_BYTE_ARRAY, baContext, ulContextLen },
6051 { CK_SP800_108_DKM_LENGTH, dkmFormat, sizeof(dkmFormat) }
6052 };
6053
6054 CK_SP800_108_KDF_PARAMS kdfParams =
6055 {
6056 CKM_AES_CMAC,
6057 DIM(dataParams),
6058 &dataParams,
6059 0, /* no addition derived keys */
6060 NULL /* no addition derived keys */
6061 };
6062
6063 CK_MECHANISM = mechanism
6064 {
6065 CKM_SP800_108_DOUBLE_PIPELINE_KDF,
6066 &kdfParams,
6067 sizeof(kdfParams)
6068 };
6069
6070 hBaseKey = GetBaseKeyHandle(.....);

```

```

6071
6072 rv = C_DeriveKey(
6073 hSession,
6074 &mechanism,
6075 hBaseKey,
6076 &derivedKeyTemplate,
6077 DIM(derivedKeyTemplate),
6078 &hDerivedKey);

```

## 6079 2.43 Miscellaneous simple key derivation mechanisms

6080 *Table 167, Miscellaneous simple key derivation Mechanisms vs. Functions*

| Mechanism                     | Functions               |                     |                            |        |                             |                     |        |
|-------------------------------|-------------------------|---------------------|----------------------------|--------|-----------------------------|---------------------|--------|
|                               | Encrypt<br>&<br>Decrypt | Sign<br>&<br>Verify | SR<br>&<br>VR <sup>1</sup> | Digest | Gen.<br>Key/<br>Key<br>Pair | Wrap<br>&<br>Unwrap | Derive |
| CKM_CONCATENATE_BASE_AND_KEY  |                         |                     |                            |        |                             |                     | ✓      |
| CKM_CONCATENATE_BASE_AND_DATA |                         |                     |                            |        |                             |                     | ✓      |
| CKM_CONCATENATE_DATA_AND_BASE |                         |                     |                            |        |                             |                     | ✓      |
| CKM_XOR_BASE_AND_DATA         |                         |                     |                            |        |                             |                     | ✓      |
| CKM_EXTRACT_KEY_FROM_KEY      |                         |                     |                            |        |                             |                     | ✓      |

### 6081 2.43.1 Definitions

6082 Mechanisms:

```

6083 CKM_CONCATENATE_BASE_AND_DATA
6084 CKM_CONCATENATE_DATA_AND_BASE
6085 CKM_XOR_BASE_AND_DATA
6086 CKM_EXTRACT_KEY_FROM_KEY
6087 CKM_CONCATENATE_BASE_AND_KEY

```

### 6088 2.43.2 Parameters for miscellaneous simple key derivation mechanisms

- 6089 ◆ **CK\_KEY\_DERIVATION\_STRING\_DATA;**
- 6090      **CK\_KEY\_DERIVATION\_STRING\_DATA\_PTR**

6091 CK\_KEY\_DERIVATION\_STRING\_DATA provides the parameters for the  
6092 CKM\_CONCATENATE\_BASE\_AND\_DATA, CKM\_CONCATENATE\_DATA\_AND\_BASE, and  
6093 CKM\_XOR\_BASE\_AND\_DATA mechanisms. It is defined as follows:

```

6094 typedef struct CK_KEY_DERIVATION_STRING_DATA {
6095 CK_BYTE_PTR pData;
6096 CK_ULONG ulLen;
6097 } CK_KEY_DERIVATION_STRING_DATA;
6098

```

6099 The fields of the structure have the following meanings:

6100      *pData*      *pointer to the byte string*

6101      *ulLen*      *length of the byte string*

6102   **CK\_KEY\_DERIVATION\_STRING\_DATA\_PTR** is a pointer to a  
6103   **CK\_KEY\_DERIVATION\_STRING\_DATA**.

6104   ◆ **CK\_EXTRACT\_PARAMS; CK\_EXTRACT\_PARAMS\_PTR**

6105   **CK\_EXTRACT\_PARAMS** provides the parameter to the **CKM\_EXTRACT\_KEY\_FROM\_KEY**  
6106   mechanism. It specifies which bit of the base key should be used as the first bit of the derived key. It is  
6107   defined as follows:

6108       typedef CK ULONG CK\_EXTRACT\_PARAMS;

6109

6110   **CK\_EXTRACT\_PARAMS\_PTR** is a pointer to a **CK\_EXTRACT\_PARAMS**.

### 6111   2.43.3 Concatenation of a base key and another key

6112   This mechanism, denoted **CKM\_CONCATENATE\_BASE\_AND\_KEY**, derives a secret key from the  
6113   concatenation of two existing secret keys. The two keys are specified by handles; the values of the keys  
6114   specified are concatenated together in a buffer.

6115   This mechanism takes a parameter, a **CK\_OBJECT\_HANDLE**. This handle produces the key value  
6116   information which is appended to the end of the base key's value information (the base key is the key  
6117   whose handle is supplied as an argument to **C\_DeriveKey**).

6118   For example, if the value of the base key is 0x01234567, and the value of the other key is 0x89ABCDEF,  
6119   then the value of the derived key will be taken from a buffer containing the string 0x0123456789ABCDEF.

- 6120   • If no length or key type is provided in the template, then the key produced by this mechanism will be a  
6121   generic secret key. Its length will be equal to the sum of the lengths of the values of the two original  
6122   keys.
- 6123   • If no key type is provided in the template, but a length is, then the key produced by this mechanism  
6124   will be a generic secret key of the specified length.
- 6125   • If no length is provided in the template, but a key type is, then that key type must have a well-defined  
6126   length. If it does, then the key produced by this mechanism will be of the type specified in the  
6127   template. If it doesn't, an error will be returned.
- 6128   • If both a key type and a length are provided in the template, the length must be compatible with that  
6129   key type. The key produced by this mechanism will be of the specified type and length.

6130   If a DES, DES2, DES3, or CDMF key is derived with this mechanism, the parity bits of the key will be set  
6131   properly.

6132   If the requested type of key requires more bytes than are available by concatenating the two original keys'  
6133   values, an error is generated.

6134   This mechanism has the following rules about key sensitivity and extractability:

- 6135   • If either of the two original keys has its **CKA\_SENSITIVE** attribute set to CK\_TRUE, so does the  
6136   derived key. If not, then the derived key's **CKA\_SENSITIVE** attribute is set either from the supplied  
6137   template or from a default value.
- 6138   • Similarly, if either of the two original keys has its **CKA\_EXTRACTABLE** attribute set to CK\_FALSE,  
6139   so does the derived key. If not, then the derived key's **CKA\_EXTRACTABLE** attribute is set either  
6140   from the supplied template or from a default value.
- 6141   • The derived key's **CKA\_ALWAYS\_SENSITIVE** attribute is set to CK\_TRUE if and only if both of the  
6142   original keys have their **CKA\_ALWAYS\_SENSITIVE** attributes set to CK\_TRUE.
- 6143   • Similarly, the derived key's **CKA\_NEVER\_EXTRACTABLE** attribute is set to CK\_TRUE if and only if  
6144   both of the original keys have their **CKA\_NEVER\_EXTRACTABLE** attributes set to CK\_TRUE.

#### 2.43.4 Concatenation of a base key and data

This mechanism, denoted **CKM\_CONCATENATE\_BASE\_AND\_DATA**, derives a secret key by concatenating data onto the end of a specified secret key.

This mechanism takes a parameter, a **CK\_KEY\_DERIVATION\_STRING\_DATA** structure, which specifies the length and value of the data which will be appended to the base key to derive another key.

For example, if the value of the base key is 0x01234567, and the value of the data is 0x89ABCDEF, then the value of the derived key will be taken from a buffer containing the string 0x0123456789ABCDEF.

- If no length or key type is provided in the template, then the key produced by this mechanism will be a generic secret key. Its length will be equal to the sum of the lengths of the value of the original key and the data.
- If no key type is provided in the template, but a length is, then the key produced by this mechanism will be a generic secret key of the specified length.
- If no length is provided in the template, but a key type is, then that key type must have a well-defined length. If it does, then the key produced by this mechanism will be of the type specified in the template. If it doesn't, an error will be returned.
- If both a key type and a length are provided in the template, the length must be compatible with that key type. The key produced by this mechanism will be of the specified type and length.

If a DES, DES2, DES3, or CDMF key is derived with this mechanism, the parity bits of the key will be set properly.

If the requested type of key requires more bytes than are available by concatenating the original key's value and the data, an error is generated.

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

- If the base key has its **CKA\_SENSITIVE** attribute set to CK\_TRUE, so does the derived key. If not, then the derived key's **CKA\_SENSITIVE** attribute is set either from the supplied template or from a default value.
- Similarly, if the base key has its **CKA\_EXTRACTABLE** attribute set to CK\_FALSE, so does the derived key. If not, then the derived key's **CKA\_EXTRACTABLE** attribute is set either from the supplied template or from a default value.
- The derived key's **CKA\_ALWAYS\_SENSITIVE** attribute is set to CK\_TRUE if and only if the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE.
- Similarly, the derived key's **CKA\_NEVER\_EXTRACTABLE** attribute is set to CK\_TRUE if and only if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE.

#### 2.43.5 Concatenation of data and a base key

This mechanism, denoted **CKM\_CONCATENATE\_DATA\_AND\_BASE**, derives a secret key by prepending data to the start of a specified secret key.

This mechanism takes a parameter, a **CK\_KEY\_DERIVATION\_STRING\_DATA** structure, which specifies the length and value of the data which will be prepended to the base key to derive another key.

For example, if the value of the base key is 0x01234567, and the value of the data is 0x89ABCDEF, then the value of the derived key will be taken from a buffer containing the string 0x89ABCDEF01234567.

- If no length or key type is provided in the template, then the key produced by this mechanism will be a generic secret key. Its length will be equal to the sum of the lengths of the data and the value of the original key.
- If no key type is provided in the template, but a length is, then the key produced by this mechanism will be a generic secret key of the specified length.
- If no length is provided in the template, but a key type is, then that key type must have a well-defined length. If it does, then the key produced by this mechanism will be of the type specified in the template. If it doesn't, an error will be returned.

- 6192 • If both a key type and a length are provided in the template, the length must be compatible with that  
6193 key type. The key produced by this mechanism will be of the specified type and length.

6194 If a DES, DES2, DES3, or CDMF key is derived with this mechanism, the parity bits of the key will be set  
6195 properly.

6196 If the requested type of key requires more bytes than are available by concatenating the data and the  
6197 original key's value, an error is generated.

6198 This mechanism has the following rules about key sensitivity and extractability:

- 6199 • If the base key has its **CKA\_SENSITIVE** attribute set to CK\_TRUE, so does the derived key. If not,  
6200 then the derived key's **CKA\_SENSITIVE** attribute is set either from the supplied template or from a  
6201 default value.
- 6202 • Similarly, if the base key has its **CKA\_EXTRACTABLE** attribute set to CK\_FALSE, so does the  
6203 derived key. If not, then the derived key's **CKA\_EXTRACTABLE** attribute is set either from the  
6204 supplied template or from a default value.
- 6205 • The derived key's **CKA\_ALWAYS\_SENSITIVE** attribute is set to CK\_TRUE if and only if the base  
6206 key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE.
- 6207 • Similarly, the derived key's **CKA\_NEVER\_EXTRACTABLE** attribute is set to CK\_TRUE if and only if  
6208 the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE.

## 6209 **2.43.6 XORing of a key and data**

6210 XORing key derivation, denoted **CKM\_XOR\_BASE\_AND\_DATA**, is a mechanism which provides the  
6211 capability of deriving a secret key by performing a bit XORing of a key pointed to by a base key handle  
6212 and some data.

6213 This mechanism takes a parameter, a **CK\_KEY\_DERIVATION\_STRING\_DATA** structure, which  
6214 specifies the data with which to XOR the original key's value.

6215 For example, if the value of the base key is 0x01234567, and the value of the data is 0x89ABCDEF, then  
6216 the value of the derived key will be taken from a buffer containing the string 0x88888888.

- 6217 • If no length or key type is provided in the template, then the key produced by this mechanism will be a  
6218 generic secret key. Its length will be equal to the minimum of the lengths of the data and the value of  
6219 the original key.
- 6220 • If no key type is provided in the template, but a length is, then the key produced by this mechanism  
6221 will be a generic secret key of the specified length.
- 6222 • If no length is provided in the template, but a key type is, then that key type must have a well-defined  
6223 length. If it does, then the key produced by this mechanism will be of the type specified in the  
6224 template. If it doesn't, an error will be returned.
- 6225 • If both a key type and a length are provided in the template, the length must be compatible with that  
6226 key type. The key produced by this mechanism will be of the specified type and length.

6227 If a DES, DES2, DES3, or CDMF key is derived with this mechanism, the parity bits of the key will be set  
6228 properly.

6229 If the requested type of key requires more bytes than are available by taking the shorter of the data and  
6230 the original key's value, an error is generated.

6231 This mechanism has the following rules about key sensitivity and extractability:

- 6232 • If the base key has its **CKA\_SENSITIVE** attribute set to CK\_TRUE, so does the derived key. If not,  
6233 then the derived key's **CKA\_SENSITIVE** attribute is set either from the supplied template or from a  
6234 default value.
- 6235 • Similarly, if the base key has its **CKA\_EXTRACTABLE** attribute set to CK\_FALSE, so does the  
6236 derived key. If not, then the derived key's **CKA\_EXTRACTABLE** attribute is set either from the  
6237 supplied template or from a default value.
- 6238 • The derived key's **CKA\_ALWAYS\_SENSITIVE** attribute is set to CK\_TRUE if and only if the base  
6239 key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE.

- 6240 • Similarly, the derived key's **CKA\_NEVER\_EXTRACTABLE** attribute is set to CK\_TRUE if and only if  
6241 the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE.

## 6242 **2.43.7 Extraction of one key from another key**

6243 Extraction of one key from another key, denoted **CKM\_EXTRACT\_KEY\_FROM\_KEY**, is a mechanism  
6244 which provides the capability of creating one secret key from the bits of another secret key.

6245 This mechanism has a parameter, a **CK\_EXTRACT\_PARAMS**, which specifies which bit of the original  
6246 key should be used as the first bit of the newly-derived key.

6247 We give an example of how this mechanism works. Suppose a token has a secret key with the 4-byte  
6248 value 0x329F84A9. We will derive a 2-byte secret key from this key, starting at bit position 21 (i.e., the  
6249 value of the parameter to the **CKM\_EXTRACT\_KEY\_FROM\_KEY** mechanism is 21).

- 6250 1. We write the key's value in binary: 0011 0010 1001 1111 1000 0100 1010 1001. We regard this  
6251 binary string as holding the 32 bits of the key, labeled as b0, b1, ..., b31.
- 6252 2. We then extract 16 consecutive bits (i.e., 2 bytes) from this binary string, starting at bit b21. We  
6253 obtain the binary string 1001 0101 0010 0110.
- 6254 3. The value of the new key is thus 0x9526.

6255 Note that when constructing the value of the derived key, it is permissible to wrap around the end of the  
6256 binary string representing the original key's value.

6257 If the original key used in this process is sensitive, then the derived key must also be sensitive for the  
6258 derivation to succeed.

- 6259 • If no length or key type is provided in the template, then an error will be returned.
- 6260 • If no key type is provided in the template, but a length is, then the key produced by this mechanism  
6261 will be a generic secret key of the specified length.
- 6262 • If no length is provided in the template, but a key type is, then that key type must have a well-defined  
6263 length. If it does, then the key produced by this mechanism will be of the type specified in the  
6264 template. If it doesn't, an error will be returned.
- 6265 • If both a key type and a length are provided in the template, the length must be compatible with that  
6266 key type. The key produced by this mechanism will be of the specified type and length.

6267 If a DES, DES2, DESS, or CDMF key is derived with this mechanism, the parity bits of the key will be set  
6268 properly.

6269 If the requested type of key requires more bytes than the original key has, an error is generated.

6270 This mechanism has the following rules about key sensitivity and extractability:

- 6271 • If the base key has its **CKA\_SENSITIVE** attribute set to CK\_TRUE, so does the derived key. If not,  
6272 then the derived key's **CKA\_SENSITIVE** attribute is set either from the supplied template or from a  
6273 default value.
- 6274 • Similarly, if the base key has its **CKA\_EXTRACTABLE** attribute set to CK\_FALSE, so does the  
6275 derived key. If not, then the derived key's **CKA\_EXTRACTABLE** attribute is set either from the  
6276 supplied template or from a default value.
- 6277 • The derived key's **CKA\_ALWAYS\_SENSITIVE** attribute is set to CK\_TRUE if and only if the base  
6278 key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE.
- 6279 • Similarly, the derived key's **CKA\_NEVER\_EXTRACTABLE** attribute is set to CK\_TRUE if and only if  
6280 the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_TRUE.

## 6281 **2.44 CMS**

6282 *Table 168, CMS Mechanisms vs. Functions*

| Mechanism   | Functions         |               |                      |        |                    |               |        |
|-------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|             | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_CMS_SIG |                   | ✓             | ✓                    |        |                    |               |        |

## 2.44.1 Definitions

Mechanisms:

CKM\_CMS\_SIG

## 2.44.2 CMS Signature Mechanism Objects

These objects provide information relating to the CKM\_CMS\_SIG mechanism. CKM\_CMS\_SIG mechanism object attributes represent information about supported CMS signature attributes in the token. They are only present on tokens supporting the **CKM\_CMS\_SIG** mechanism, but must be present on those tokens.

Table 169, CMS Signature Mechanism Object Attributes

| Attribute                    | Data type  | Meaning                                                                                                                             |
|------------------------------|------------|-------------------------------------------------------------------------------------------------------------------------------------|
| CKA_REQUIRED_CMS_ATTRIBUTES  | Byte array | Attributes the token always will include in the set of CMS signed attributes                                                        |
| CKA_DEFAULT_CMS_ATTRIBUTES   | Byte array | Attributes the token will include in the set of CMS signed attributes in the absence of any attributes specified by the application |
| CKA_SUPPORTED_CMS_ATTRIBUTES | Byte array | Attributes the token may include in the set of CMS signed attributes upon request by the application                                |

The contents of each byte array will be a DER-encoded list of CMS **Attributes** with optional accompanying values. Any attributes in the list shall be identified with its object identifier, and any values shall be DER-encoded. The list of attributes is defined in ASN.1 as:

```

Attributes ::= SET SIZE (1..MAX) OF Attribute
Attribute ::= SEQUENCE {
 attrType OBJECT IDENTIFIER,
 attrValues SET OF ANY DEFINED BY OBJECT IDENTIFIER
 OPTIONAL
}
```

The client may not set any of the attributes.

## 2.44.3 CMS mechanism parameters

- **CK\_CMS\_SIG\_PARAMS, CK\_CMS\_SIG\_PARAMS\_PTR**

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

```

typedef struct CK_CMS_SIG_PARAMS {
 CK_OBJECT_HANDLE certificateHandle;
 CK_MECHANISM_PTR pSigningMechanism;
 CK_MECHANISM_PTR pDigestMechanism;
```

|      |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------|----------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 6310 | CK_UTF8CHAR_PTR                                          | pContentType;                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 6311 | CK_BYTE_PTR                                              | pRequestedAttributes;                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 6312 | CK ULONG                                                 | ulRequestedAttributesLen;                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 6313 | CK_BYTE_PTR                                              | pRequiredAttributes;                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 6314 | CK ULONG                                                 | ulRequiredAttributesLen;                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 6315 | }                                                        | CK_CMS_SIG_PARAMS;                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 6316 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6317 | The fields of the structure have the following meanings: |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6318 | <i>certificateHandle</i>                                 | <i>Object handle for a certificate associated with the signing key. The token may use information from this certificate to identify the signer in the <b>SignerInfo</b> result value. CertificateHandle may be NULL_PTR if the certificate is not available as a PKCS #11 object or if the calling application leaves the choice of certificate completely to the token.</i>                                                                                                                                                                                                                                                                                        |
| 6319 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6320 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6321 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6322 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6323 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6324 | <i>pSigningMechanism</i>                                 | <i>Mechanism to use when signing a constructed CMS <b>SignedAttributes</b> value. E.g. CKM_SHA1_RSA_PKCS.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 6325 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6326 | <i>pDigestMechanism</i>                                  | <i>Mechanism to use when digesting the data. Value shall be NULL_PTR when the digest mechanism to use follows from the pSigningMechanism parameter.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 6327 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6328 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6329 | <i>pContentType</i>                                      | <i>NULL-terminated string indicating complete MIME Content-type of message to be signed; or the value NULL_PTR if the message is a MIME object (which the token can parse to determine its MIME Content-type if required). Use the value "application/octet-stream" if the MIME type for the message is unknown or undefined. Note that the pContentType string shall conform to the syntax specified in RFC 2045, i.e. any parameters needed for correct presentation of the content by the token (such as, for example, a non-default "charset") must be present. The token must follow rules and procedures defined in RFC 2045 when presenting the content.</i> |
| 6330 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6331 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6332 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6333 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6334 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6335 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6336 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6337 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6338 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6339 | <i>pRequestedAttributes</i>                              | <i>Pointer to DER-encoded list of CMS <b>Attributes</b> the caller requests to be included in the signed attributes. Token may freely ignore this list or modify any supplied values.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 6340 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6341 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6342 | <i>ulRequestedAttributesLen</i>                          | <i>Length in bytes of the value pointed to by pRequestedAttributes</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 6343 | <i>pRequiredAttributes</i>                               | <i>Pointer to DER-encoded list of CMS <b>Attributes</b> (with accompanying values) required to be included in the resulting signed attributes. Token must not modify any supplied values. If the token does not support one or more of the attributes, or does not accept provided values, the signature operation will fail. The token will use its own default attributes when signing if both the pRequestedAttributes and pRequiredAttributes field are set to NULL_PTR.</i>                                                                                                                                                                                    |
| 6344 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6345 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6346 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6347 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6348 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6349 |                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 6350 | <i>ulRequiredAttributesLen</i>                           | <i>Length in bytes, of the value pointed to by pRequiredAttributes.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

#### 2.44.4 CMS signatures

The CMS mechanism, denoted **CKM\_CMS\_SIG**, is a multi-purpose mechanism based on the structures defined in PKCS #7 and RFC 2630. It supports single- or multiple-part signatures with and without message recovery. The mechanism is intended for use with, e.g., PTDs (see MeT-PTD) or other capable tokens. The token will construct a CMS **SignedAttributes** value and compute a signature on this value.

6356 The content of the **SignedAttributes** value is decided by the token, however the caller can suggest some  
6357 attributes in the parameter *pRequestedAttributes*. The caller can also require some attributes to be  
6358 present through the parameters *pRequiredAttributes*. The signature is computed in accordance with the  
6359 parameter *pSigningMechanism*.

6360 When this mechanism is used in successful calls to **C\_Sign** or **C\_SignFinal**, the *pSignature* return value  
6361 will point to a DER-encoded value of type **SignerInfo**. **SignerInfo** is defined in ASN.1 as follows (for a  
6362 complete definition of all fields and types, see RFC 2630):

```
6363 SignerInfo ::= SEQUENCE {
6364 version CMSVersion,
6365 sid SignerIdentifier,
6366 digestAlgorithm DigestAlgorithmIdentifier,
6367 signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
6368 signatureAlgorithm SignatureAlgorithmIdentifier,
6369 signature SignatureValue,
6370 unsignedAttrs [1] IMPLICIT UnsignedAttributes
6371 OPTIONAL }
```

6372 The *certificateHandle* parameter, when set, helps the token populate the **sid** field of the **SignerInfo** value.  
6373 If *certificateHandle* is NULL\_PTR the choice of a suitable certificate reference in the **SignerInfo** result  
6374 value is left to the token (the token could, e.g., interact with the user).

6375 This mechanism shall not be used in calls to **C\_Verify** or **C\_VerifyFinal** (use the *pSigningMechanism*  
6376 mechanism instead).

6377 For the *pRequiredAttributes* field, the token may have to interact with the user to find out whether to  
6378 accept a proposed value or not. The token should never accept any proposed attribute values without  
6379 some kind of confirmation from its owner (but this could be through, e.g., configuration or policy settings  
6380 and not direct interaction). If a user rejects proposed values, or the signature request as such, the value  
6381 CKR\_FUNCTION\_REJECTED shall be returned.

6382 When possible, applications should use the **CKM\_CMS\_SIG** mechanism when generating CMS-  
6383 compatible signatures rather than lower-level mechanisms such as **CKM\_SHA1\_RSA\_PKCS**. This is  
6384 especially true when the signatures are to be made on content that the token is able to present to a user.  
6385 Exceptions may include those cases where the token does not support a particular signing attribute. Note  
6386 however that the token may refuse usage of a particular signature key unless the content to be signed is  
6387 known (i.e. the **CKM\_CMS\_SIG** mechanism is used).

6388 When a token does not have presentation capabilities, the PKCS #11-aware application may avoid  
6389 sending the whole message to the token by electing to use a suitable signature mechanism (e.g.  
6390 **CKM\_RSA\_PKCS**) as the *pSigningMechanism* value in the **CK\_CMS\_SIG\_PARAMS** structure, and  
6391 digesting the message itself before passing it to the token.

6392 PKCS #11-aware applications making use of tokens with presentation capabilities, should attempt to  
6393 provide messages to be signed by the token in a format possible for the token to present to the user.  
6394 Tokens that receive multipart MIME-messages for which only certain parts are possible to present may  
6395 fail the signature operation with a return value of **CKR\_DATA\_INVALID**, but may also choose to add a  
6396 signing attribute indicating which parts of the message were possible to present.

## 6397 2.45 Blowfish

6398 Blowfish, a secret-key block cipher. It is a Feistel network, iterating a simple encryption function 16 times.  
6399 The block size is 64 bits, and the key can be any length up to 448 bits. Although there is a complex  
6400 initialization phase required before any encryption can take place, the actual encryption of data is very  
6401 efficient on large microprocessors.

6402

6403 *Table 170, Blowfish Mechanisms vs. Functions*

| Mechanism            | Functions         |               |                      |        |                    |               |        |
|----------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                      | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_BLOWFISH_CBC     | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_BLOWFISH_CBC_PAD | ✓                 |               |                      |        |                    | ✓             |        |

## 6404 2.45.1 Definitions

6405 This section defines the key type "CKK\_BLOWFISH" for type CK\_KEY\_TYPE as used in the  
 6406 CKA\_KEY\_TYPE attribute of key objects.

6407 Mechanisms:

6408     CKM\_BLOWFISH\_KEY\_GEN  
 6409     CKM\_BLOWFISH\_CBC  
 6410     CKM\_BLOWFISH\_CBC\_PAD

## 6411 2.45.2 BLOWFISH secret key objects

6412 Blowfish secret key objects (object class CKO\_SECRET\_KEY, key type CKK\_BLOWFISH) hold Blowfish  
 6413 keys. The following table defines the Blowfish secret key object attributes, in addition to the common  
 6414 attributes defined for this object class:

6415 *Table 171, BLOWFISH Secret Key Object*

| Attribute                    | Data type  | Meaning                                                                                    |
|------------------------------|------------|--------------------------------------------------------------------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup> | Byte array | Key value the key can be any length up to 448 bits. Bit length restricted to a byte array. |
| CKA_VALUE_LEN <sup>2,3</sup> | CK ULONG   | Length in bytes of key value                                                               |

6416 - Refer to [PKCS11-Base] table 11 for footnotes

6417 The following is a sample template for creating an Blowfish secret key object:

```
6418 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
6419 CK_KEY_TYPE keyType = CKK_BLOWFISH;
6420 CK_UTF8CHAR label[] = "A blowfish secret key object";
6421 CK_BYTE value[16] = {...};
6422 CK_BBOOL true = CK_TRUE;
6423 CK_ATTRIBUTE template[] = {
6424 {CKA_CLASS, &class, sizeof(class)},
6425 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
6426 {CKA_TOKEN, &true, sizeof(true)},
6427 {CKA_LABEL, label, sizeof(label)-1},
6428 {CKA_ENCRYPT, &true, sizeof(true)},
6429 {CKA_VALUE, value, sizeof(value)}}
6430 };
```

### 6431 2.45.3 Blowfish key generation

6432 The Blowfish key generation mechanism, denoted **CKM\_BLOWFISH\_KEY\_GEN**, is a key generation  
6433 mechanism Blowfish.

6434 It does not have a parameter.

6435 The mechanism generates Blowfish keys with a particular length, as specified in the **CKA\_VALUE\_LEN**  
6436 attribute of the template for the key.

6437 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
6438 key. Other attributes supported by the key type (specifically, the flags indicating which functions the key  
6439 supports) may be specified in the template for the key, or else are assigned default initial values.

6440 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
6441 specify the supported range of key sizes in bytes.

### 6442 2.45.4 Blowfish-CBC

6443 Blowfish-CBC, denoted **CKM\_BLOWFISH\_CBC**, is a mechanism for single- and multiple-part encryption  
6444 and decryption; key wrapping; and key unwrapping.

6445 It has a parameter, a 8-byte initialization vector.

6446 This mechanism can wrap and unwrap any secret key. For wrapping, the mechanism encrypts the value  
6447 of the **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to block size  
6448 minus one null bytes so that the resulting length is a multiple of the block size. The output data is the  
6449 same length as the padded input data. It does not wrap the key type, key length, or any other information  
6450 about the key; the application must convey these separately.

6451 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
6452 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the  
6453 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
6454 attribute of the new key; other attributes required by the key type must be specified in the template.

6455 Constraints on key types and the length of data are summarized in the following table:

6456 Table 172, BLOWFISH-CBC: Key and Data Length

| Function    | Key type | Input Length           | Output Length                                                           |
|-------------|----------|------------------------|-------------------------------------------------------------------------|
| C_Encrypt   | BLOWFISH | Multiple of block size | Same as input length                                                    |
| C_Decrypt   | BLOWFISH | Multiple of block size | Same as input length                                                    |
| C_WrapKey   | BLOWFISH | Any                    | Input length rounded up to<br>multiple of the block size                |
| C_UnwrapKey | BLOWFISH | Multiple of block size | Determined by type of key<br>being unwrapped or<br><b>CKA_VALUE_LEN</b> |

6457 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
6458 specify the supported range of BLOWFISH key sizes, in bytes.

### 6459 2.45.5 Blowfish-CBC with PKCS padding

6460 Blowfish-CBC-PAD, denoted **CKM\_BLOWFISH\_CBC\_PAD**, is a mechanism for single- and multiple-part  
6461 encryption and decryption, key wrapping and key unwrapping, cipher-block chaining mode and the block  
6462 cipher padding method detailed in PKCS #7.

6463 It has a parameter, a 8-byte initialization vector.

6464 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the  
6465 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified for  
6466 the **CKA\_VALUE\_LEN** attribute.

6467 The entries in the table below for data length constraints when wrapping and unwrapping keys do not  
6468 apply to wrapping and unwrapping private keys.

6469 Constraints on key types and the length of data are summarized in the following table:

6470

6471 *Table 173, BLOWFISH-CBC with PKCS Padding: Key and Data Length*

| Function    | Key type | Input Length           | Output Length                                                         |
|-------------|----------|------------------------|-----------------------------------------------------------------------|
| C_Encrypt   | BLOWFISH | Any                    | Input length rounded up to multiple of the block size                 |
| C_Decrypt   | BLOWFISH | Multiple of block size | Between 1 and block length block size bytes shorter than input length |
| C_WrapKey   | BLOWFISH | Any                    | Input length rounded up to multiple of the block size                 |
| C_UnwrapKey | BLOWFISH | Multiple of block size | Between 1 and block length block size bytes shorter than input length |

## 6472 **2.46 Twofish**

6473 Ref. <https://www.schneier.com/twofish.html>

### 6474 **2.46.1 Definitions**

6475 This section defines the key type “CKK\_TWOFISH” for type CK\_KEY\_TYPE as used in the  
6476 CKA\_KEY\_TYPE attribute of key objects.

6477 Mechanisms:

6478       CKM\_TWOFISH\_KEY\_GEN  
6479       CKM\_TWOFISH\_CBC  
6480       CKM\_TWOFISH\_CBC\_PAD

6481

### 6482 **2.46.2 Twofish secret key objects**

6483 Twofish secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_TWOFISH**) hold Twofish  
6484 keys. The following table defines the Twofish secret key object attributes, in addition to the common  
6485 attributes defined for this object class:

6486 *Table 174, Twofish Secret Key Object*

| Attribute                    | Data type  | Meaning                              |
|------------------------------|------------|--------------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup> | Byte array | Key value 128-, 192-, or 256-bit key |
| CKA_VALUE_LEN <sup>2,3</sup> | CK ULONG   | Length in bytes of key value         |

6487 - Refer to [PKCS11-Base] table 11 for footnotes

6488 The following is a sample template for creating an TWOFISH secret key object:

```
6489 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
6490 CK_KEY_TYPE keyType = CKK_TWOFISH;
6491 CK_UTF8CHAR label[] = "A twofish secret key object";
6492 CK_BYTE value[16] = {...};
6493 CK_BBOOL true = CK_TRUE;
6494 CK_ATTRIBUTE template[] = {
6495 {CKA_CLASS, &class, sizeof(class)},
```

```

6496 { CKA_KEY_TYPE, &keyType, sizeof(keyType) },
6497 { CKA_TOKEN, &true, sizeof(true) },
6498 { CKA_LABEL, label, sizeof(label)-1 },
6499 { CKA_ENCRYPT, &true, sizeof(true) },
6500 { CKA_VALUE, value, sizeof(value) }
6501 } ;

```

### 6502 2.46.3 Twofish key generation

6503 The Twofish key generation mechanism, denoted **CKM\_TWOFISH\_KEY\_GEN**, is a key generation  
 6504 mechanism Twofish.

6505 It does not have a parameter.

6506 The mechanism generates Blowfish keys with a particular length, as specified in the **CKA\_VALUE\_LEN**  
 6507 attribute of the template for the key.

6508 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
 6509 key. Other attributes supported by the key type (specifically, the flags indicating which functions the key  
 6510 supports) may be specified in the template for the key, or else are assigned default initial values.

6511 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 6512 specify the supported range of key sizes, in bytes.

### 6513 2.46.4 Twofish -CBC

6514 Twofish-CBC, denoted **CKM\_TWOFISH\_CBC**, is a mechanism for single- and multiple-part encryption  
 6515 and decryption; key wrapping; and key unwrapping.

6516 It has a parameter, a 16-byte initialization vector.

### 6517 2.46.5 Twofish-CBC with PKCS padding

6518 Twofish-CBC-PAD, denoted **CKM\_TWOFISH\_CBC\_PAD**, is a mechanism for single- and multiple-part  
 6519 encryption and decryption, key wrapping and key unwrapping, cipher-block chaining mode and the block  
 6520 cipher padding method detailed in PKCS #7.

6521 It has a parameter, a 16-byte initialization vector.

6522 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the  
 6523 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified for  
 6524 the **CKA\_VALUE\_LEN** attribute.

## 6525 2.47 CAMELLIA

6526 Camellia is a block cipher with 128-bit block size and 128-, 192-, and 256-bit keys, similar to AES.  
 6527 Camellia is described e.g. in IETF RFC 3713.

6528 *Table 175, Camellia Mechanisms vs. Functions*

| Mechanism            | Functions         |               |                      |        |                     |               |       |
|----------------------|-------------------|---------------|----------------------|--------|---------------------|---------------|-------|
|                      | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen . Key/ Key Pair | Wrap & Unwrap | Deriv |
| CKM_CAMELLIA_KEY_GEN |                   |               |                      |        | ✓                   |               |       |
| CKM_CAMELLIA_ECB     | ✓                 |               |                      |        |                     | ✓             |       |
| CKM_CAMELLIA_CBC     | ✓                 |               |                      |        |                     | ✓             |       |

| Mechanism                     | Functions         |               |                      |        |                     |               |        |
|-------------------------------|-------------------|---------------|----------------------|--------|---------------------|---------------|--------|
|                               | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen . Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_CAMELLIA_CBC_PAD          | ✓                 |               |                      |        |                     | ✓             |        |
| CKM_CAMELLIA_MAC_GENERAL      |                   | ✓             |                      |        |                     |               |        |
| CKM_CAMELLIA_MAC              |                   | ✓             |                      |        |                     |               |        |
| CKM_CAMELLIA_ECB_ENCRYPT_DATA |                   |               |                      |        |                     |               | ✓      |
| CKM_CAMELLIA_CBC_ENCRYPT_DATA |                   |               |                      |        |                     |               | ✓      |

## 2.47.1 Definitions

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

Mechanisms:

- CKM\_CAMELLIA\_KEY\_GEN
- CKM\_CAMELLIA\_ECB
- CKM\_CAMELLIA\_CBC
- CKM\_CAMELLIA\_MAC
- CKM\_CAMELLIA\_MAC\_GENERAL
- CKM\_CAMELLIA\_CBC\_PAD

## 2.47.2 Camellia secret key objects

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

Table 176, Camellia Secret Key Object Attributes

| Attribute                      | Data type  | Meaning                         |
|--------------------------------|------------|---------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup>   | Byte array | Key value (16, 24, or 32 bytes) |
| CKA_VALUE_LEN <sup>2,3,6</sup> | CK ULONG   | Length in bytes of key value    |

- Refer to [PKCS11-Base] table 11 for footnotes.

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

```

6546 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
6547 CK_KEY_TYPE keyType = CKK_CAMELLIA;
6548 CK_UTF8CHAR label[] = "A Camellia secret key object";
6549 CK_BYTE value[] = { ... };
6550 CK_BBOOL true = CK_TRUE;
6551 CK_ATTRIBUTE template[] = {
6552 {CKA_CLASS, &class, sizeof(class)},
6553 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
```

```

6554 { CKA_TOKEN, &true, sizeof(true) },
6555 { CKA_LABEL, label, sizeof(label)-1 },
6556 { CKA_ENCRYPT, &true, sizeof(true) },
6557 { CKA_VALUE, value, sizeof(value) }
6558 };

```

### 2.47.3 Camellia key generation

6560 The Camellia key generation mechanism, denoted CKM\_CAMELLIA\_KEY\_GEN, is a key generation  
 6561 mechanism for Camellia.

6562 It does not have a parameter.

6563 The mechanism generates Camellia keys with a particular length in bytes, as specified in the  
 6564 **CKA\_VALUE\_LEN** attribute of the template for the key.

6565 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
 6566 key. Other attributes supported by the Camellia key type (specifically, the flags indicating which functions  
 6567 the key supports) may be specified in the template for the key, or else are assigned default initial values.

6568 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 6569 specify the supported range of Camellia key sizes, in bytes.

### 2.47.4 Camellia-ECB

6571 Camellia-ECB, denoted **CKM\_CAMELLIA\_ECB**, is a mechanism for single- and multiple-part encryption  
 6572 and decryption; key wrapping; and key unwrapping, based on Camellia and electronic codebook mode.

6573 It does not have a parameter.

6574 This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to  
 6575 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 block size minus  
 6577 one null bytes so that the resulting length is a multiple of the block size. The output data is the same  
 6578 length as the padded input data. It does not wrap the key type, key length, or any other information about  
 6579 the key; the application must convey these separately.

6580 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
 6581 **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**  
 6583 attribute of the new key; other attributes required by the key type must be specified in the template.

6584 Constraints on key types and the length of data are summarized in the following table:

6585 *Table 177, Camellia-ECB: Key and Data Length*

| Function    | Key type     | Input length           | Output length                                                     | Comments      |
|-------------|--------------|------------------------|-------------------------------------------------------------------|---------------|
| C_Encrypt   | CKK_CAMELLIA | multiple of block size | same as input length                                              | no final part |
| C_Decrypt   | CKK_CAMELLIA | multiple of block size | same as input length                                              | no final part |
| C_WrapKey   | CKK_CAMELLIA | any                    | input length rounded up to multiple of block size                 |               |
| C_UnwrapKey | CKK_CAMELLIA | multiple of block size | determined by type of key being unwrapped or <b>CKA_VALUE_LEN</b> |               |

6586 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 6587 specify the supported range of Camellia key sizes, in bytes.

## 6588 2.47.5 Camellia-CBC

6589 Camellia-CBC, denoted **CKM\_CAMELLIA\_CBC**, is a mechanism for single- and multiple-part encryption  
6590 and decryption; key wrapping; and key unwrapping, based on Camellia and cipher-block chaining mode.

6591 It has a parameter, a 16-byte initialization vector.

6592 This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to  
6593 wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the  
6594 **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to block size minus  
6595 one null bytes so that the resulting length is a multiple of the block size. The output data is the same  
6596 length as the padded input data. It does not wrap the key type, key length, or any other information about  
6597 the key; the application must convey these separately.

6598 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
6599 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the  
6600 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
6601 attribute of the new key; other attributes required by the key type must be specified in the template.

6602 Constraints on key types and the length of data are summarized in the following table:

6603 *Table 178, Camellia-CBC: Key and Data Length*

| Function    | Key type     | Input length           | Output length                                                     | Comments      |
|-------------|--------------|------------------------|-------------------------------------------------------------------|---------------|
| C_Encrypt   | CKK_CAMELLIA | multiple of block size | same as input length                                              | no final part |
| C_Decrypt   | CKK_CAMELLIA | multiple of block size | same as input length                                              | no final part |
| C_WrapKey   | CKK_CAMELLIA | any                    | input length rounded up to multiple of the block size             |               |
| C_UnwrapKey | CKK_CAMELLIA | multiple of block size | determined by type of key being unwrapped or <b>CKA_VALUE_LEN</b> |               |

6604 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
6605 specify the supported range of Camellia key sizes, in bytes.

## 6606 2.47.6 Camellia-CBC with PKCS padding

6607 Camellia-CBC with PKCS padding, denoted **CKM\_CAMELLIA\_CBC\_PAD**, is a mechanism for single-  
6608 and multiple-part encryption and decryption; key wrapping; and key unwrapping, based on Camellia;  
6609 cipher-block chaining mode; and the block cipher padding method detailed in PKCS #7.

6610 It has a parameter, a 16-byte initialization vector.

6611 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the  
6612 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified  
6613 for the **CKA\_VALUE\_LEN** attribute.

6614 In addition to being able to wrap and unwrap secret keys, this mechanism can wrap and unwrap RSA,  
6615 Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see Section  
6616 TBA for details). The entries in the table below for data length constraints when wrapping and  
6617 unwrapping keys do not apply to wrapping and unwrapping private keys.

6618 Constraints on key types and the length of data are summarized in the following table:

6619 Table 179, Camellia-CBC with PKCS Padding: Key and Data Length

| Function    | Key type     | Input length           | Output length                                              |
|-------------|--------------|------------------------|------------------------------------------------------------|
| C_Encrypt   | CKK_CAMELLIA | any                    | input length rounded up to multiple of the block size      |
| C_Decrypt   | CKK_CAMELLIA | multiple of block size | between 1 and block size bytes shorter than input length   |
| C_WrapKey   | CKK_CAMELLIA | any                    | input length rounded up to multiple of the block size      |
| C_UnwrapKey | CKK_CAMELLIA | multiple of block size | between 1 and block length bytes shorter than input length |

6620 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 6621 specify the supported range of Camellia key sizes, in bytes.

6622

## 6623 2.47.7 CAMELLIA with Counter mechanism parameters

### 6624 ♦ CK\_CAMELLIA\_CTR\_PARAMS; CK\_CAMELLIA\_CTR\_PARAMS\_PTR

6625 **CK\_CAMELLIA\_CTR\_PARAMS** is a structure that provides the parameters to the  
 6626 **CKM\_CAMELLIA\_CTR** mechanism. It is defined as follows:

```
6627 typedef struct CK_CAMELLIA_CTR_PARAMS {
6628 CK ULONG ulCounterBits;
6629 CK_BYTE cb[16];
6630 } CK_CAMELLIA_CTR_PARAMS;
```

6632 *ulCounterBits* specifies the number of bits in the counter block (*cb*) that shall be incremented. This  
 6633 number shall be such that  $0 < \text{ulCounterBits} \leq 128$ . For any values outside this range the mechanism  
 6634 shall return **CKR\_MECHANISM\_PARAM\_INVALID**.

6635 It's up to the caller to initialize all of the bits in the counter block including the counter bits. The counter  
 6636 bits are the least significant bits of the counter block (*cb*). They are a big-endian value usually starting  
 6637 with 1. The rest of '*cb*' is for the nonce, and maybe an optional IV.

6638 E.g. as defined in [RFC 3686]:

```
6639 0 1 2 3
6640 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
6641 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
6642 | Nonce |
6643 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
6644 | Initialization Vector (IV) |
6645 |
6646 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
6647 | Block Counter |
6648 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

6649  
 6650 This construction permits each packet to consist of up to  $2^{32}-1$  blocks = 4,294,967,295 blocks =  
 6651 68,719,476,720 octets.

6652 **CK\_CAMELLIA\_CTR\_PARAMS\_PTR** is a pointer to a **CK\_CAMELLIA\_CTR\_PARAMS**.

6653

## 6654 2.47.8 General-length Camellia-MAC

6655 General-length Camellia -MAC, denoted CKM\_CAMELLIA\_MAC\_GENERAL, is a mechanism for single-  
6656 and multiple-part signatures and verification, based on Camellia and data authentication as defined  
6657 in.[CAMELLIA]

6658 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS** structure, which specifies the output length  
6659 desired from the mechanism.

6660 The output bytes from this mechanism are taken from the start of the final Camellia cipher block produced  
6661 in the MACing process.

6662 Constraints on key types and the length of data are summarized in the following table:

6663 *Table 180, General-length Camellia-MAC: Key and Data Length*

| Function | Key type     | Data length | Signature length                         |
|----------|--------------|-------------|------------------------------------------|
| C_Sign   | CKK_CAMELLIA | any         | 1-block size, as specified in parameters |
| C_Verify | CKK_CAMELLIA | any         | 1-block size, as specified in parameters |

6664 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
6665 specify the supported range of Camellia key sizes, in bytes.

## 6666 2.47.9 Camellia-MAC

6667 Camellia-MAC, denoted by **CKM\_CAMELLIA\_MAC**, is a special case of the general-length Camellia-  
6668 MAC mechanism. Camellia-MAC always produces and verifies MACs that are half the block size in  
6669 length.

6670 It does not have a parameter.

6671 Constraints on key types and the length of data are summarized in the following table:

6672 *Table 181, Camellia-MAC: Key and Data Length*

| Function | Key type     | Data length | Signature length       |
|----------|--------------|-------------|------------------------|
| C_Sign   | CKK_CAMELLIA | any         | ½ block size (8 bytes) |
| C_Verify | CKK_CAMELLIA | any         | ½ block size (8 bytes) |

6673 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
6674 specify the supported range of Camellia key sizes, in bytes.

## 6675 2.48 Key derivation by data encryption - Camellia

6676 These mechanisms allow derivation of keys using the result of an encryption operation as the key value.  
6677 They are for use with the C\_DeriveKey function.

### 6678 2.48.1 Definitions

6679 Mechanisms:

6680       **CKM\_CAMELLIA\_ECB\_ENCRYPT\_DATA**

6681       **CKM\_CAMELLIA\_CBC\_ENCRYPT\_DATA**

6682

```
6683 typedef struct CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS {
6684 CK_BYTE iv[16];
6685 CK_BYTE_PTR pData;
6686 CK ULONG length;
6687 } CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS;
```

6688  
6689     typedef CK\_CAMELLIA\_CBC\_ENCRYPT\_DATA\_PARAMS CK\_PTR  
6690                 CK\_CAMELLIA\_CBC\_ENCRYPT\_DATA\_PARAMS\_PTR;

## 6691 2.48.2 Mechanism Parameters

6692 Uses CK\_CAMELLIA\_CBC\_ENCRYPT\_DATA\_PARAMS, and CK\_KEY\_DERIVATION\_STRING\_DATA.

6693 *Table 182, Mechanism Parameters for Camellia-based key derivation*

|                               |                                                                                                                                                           |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKM_CAMELLIA_ECB_ENCRYPT_DATA | Uses CK_KEY_DERIVATION_STRING_DATA structure. Parameter is the data to be encrypted and must be a multiple of 16 long.                                    |
| CKM_CAMELLIA_CBC_ENCRYPT_DATA | Uses CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS. Parameter is an 16 byte IV value followed by the data. The data value part must be a multiple of 16 bytes long. |

6694

## 6695 2.49 ARIA

6696 ARIA is a block cipher with 128-bit block size and 128-, 192-, and 256-bit keys, similar to AES. ARIA is  
6697 described in NSRI “Specification of ARIA”.

6698 *Table 183, ARIA Mechanisms vs. Functions*

| Mechanism                 | Functions         |               |                      |        |                    |               |        |
|---------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                           | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_ARIA_KEY_GEN          |                   |               |                      |        | ✓                  |               |        |
| CKM_ARIA_ECB              | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_ARIA_CBC              | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_ARIA_CBC_PAD          | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_ARIA_MAC_GENERAL      |                   | ✓             |                      |        |                    |               |        |
| CKM_ARIA_MAC              |                   | ✓             |                      |        |                    |               |        |
| CKM_ARIA_ECB_ENCRYPT_DATA |                   |               |                      |        |                    |               | ✓      |
| CKM_ARIA_CBC_ENCRYPT_DATA |                   |               |                      |        |                    |               | ✓      |

## 6699 2.49.1 Definitions

6700 This section defines the key type “CKK\_ARIA” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE  
6701 attribute of key objects.

6702 Mechanisms:

6703     CKM\_ARIA\_KEY\_GEN  
6704     CKM\_ARIA\_ECB  
6705     CKM\_ARIA\_CBC  
6706     CKM\_ARIA\_MAC  
6707     CKM\_ARIA\_MAC\_GENERAL

6708 CKM\_ARIA\_CBC\_PAD

## 6709 2.49.2 Aria secret key objects

6710 ARIA secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_ARIA**) hold ARIA keys. The  
6711 following table defines the ARIA secret key object attributes, in addition to the common attributes defined  
6712 for this object class:

6713 *Table 184, ARIA Secret Key Object Attributes*

| Attribute                      | Data type  | Meaning                         |
|--------------------------------|------------|---------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup>   | Byte array | Key value (16, 24, or 32 bytes) |
| CKA_VALUE_LEN <sup>2,3,6</sup> | CK ULONG   | Length in bytes of key value    |

6714 - Refer to [PKCS11-Base] table 11 for footnotes.

6715 The following is a sample template for creating an ARIA secret key object:

```
6716 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
6717 CK_KEY_TYPE keyType = CKK_ARIA;
6718 CK_UTF8CHAR label[] = "An ARIA secret key object";
6719 CK_BYTE value[] = { ... };
6720 CK_BBOOL true = CK_TRUE;
6721 CK_ATTRIBUTE template[] = {
6722 { CKA_CLASS, &class, sizeof(class) },
6723 { CKA_KEY_TYPE, &keyType, sizeof(keyType) },
6724 { CKA_TOKEN, &true, sizeof(true) },
6725 { CKA_LABEL, label, sizeof(label)-1 },
6726 { CKA_ENCRYPT, &true, sizeof(true) },
6727 { CKA_VALUE, value, sizeof(value) }
6728 };
```

## 6729 2.49.3 ARIA key generation

6730 The ARIA key generation mechanism, denoted CKM\_ARIA\_KEY\_GEN, is a key generation mechanism  
6731 for Aria.

6732 It does not have a parameter.

6733 The mechanism generates ARIA keys with a particular length in bytes, as specified in the  
6734 **CKA\_VALUE\_LEN** attribute of the template for the key.

6735 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
6736 key. Other attributes supported by the ARIA key type (specifically, the flags indicating which functions the  
6737 key supports) may be specified in the template for the key, or else are assigned default initial values.

6738 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
6739 specify the supported range of ARIA key sizes, in bytes.

## 6740 2.49.4 ARIA-ECB

6741 ARIA-ECB, denoted **CKM\_ARIA\_ECB**, is a mechanism for single- and multiple-part encryption and  
6742 decryption; key wrapping; and key unwrapping, based on Aria and electronic codebook mode.

6743 It does not have a parameter.

6744 This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to  
6745 wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the  
6746 **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to block size minus

6747 one null bytes so that the resulting length is a multiple of the block size. The output data is the same  
6748 length as the padded input data. It does not wrap the key type, key length, or any other information about  
6749 the key; the application must convey these separately.

6750 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
6751 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the  
6752 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
6753 attribute of the new key; other attributes required by the key type must be specified in the template.

6754 Constraints on key types and the length of data are summarized in the following table:

6755 *Table 185, ARIA-ECB: Key and Data Length*

| Function    | Key type | Input length           | Output length                                                     | Comments      |
|-------------|----------|------------------------|-------------------------------------------------------------------|---------------|
| C_Encrypt   | CKK_ARIA | multiple of block size | same as input length                                              | no final part |
| C_Decrypt   | CKK_ARIA | multiple of block size | same as input length                                              | no final part |
| C_WrapKey   | CKK_ARIA | any                    | input length rounded up to multiple of block size                 |               |
| C_UnwrapKey | CKK_ARIA | multiple of block size | determined by type of key being unwrapped or <b>CKA_VALUE_LEN</b> |               |

6756 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
6757 specify the supported range of ARIA key sizes, in bytes.

## 6758 2.49.5 ARIA-CBC

6759 ARIA-CBC, denoted **CKM\_ARIA\_CBC**, is a mechanism for single- and multiple-part encryption and  
6760 decryption; key wrapping; and key unwrapping, based on ARIA and cipher-block chaining mode.

6761 It has a parameter, a 16-byte initialization vector.

6762 This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to  
6763 wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the  
6764 **CKA\_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to block size minus  
6765 one null bytes so that the resulting length is a multiple of the block size. The output data is the same  
6766 length as the padded input data. It does not wrap the key type, key length, or any other information about  
6767 the key; the application must convey these separately.

6768 For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the  
6769 **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports it, the  
6770 **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the **CKA\_VALUE**  
6771 attribute of the new key; other attributes required by the key type must be specified in the template.

6772 Constraints on key types and the length of data are summarized in the following table:

6773 *Table 186, ARIA-CBC: Key and Data Length*

| Function    | Key type | Input length           | Output length                                              | Comments      |
|-------------|----------|------------------------|------------------------------------------------------------|---------------|
| C_Encrypt   | CKK_ARIA | multiple of block size | same as input length                                       | no final part |
| C_Decrypt   | CKK_ARIA | multiple of block size | same as input length                                       | no final part |
| C_WrapKey   | CKK_ARIA | any                    | input length rounded up to multiple of the block size      |               |
| C_UnwrapKey | CKK_ARIA | multiple of block size | determined by type of key being unwrapped or CKA_VALUE_LEN |               |

6774 For this mechanism, the ulMinKeySize and ulMaxKeySize fields of the CK\_MECHANISM\_INFO structure  
 6775 specify the supported range of Aria key sizes, in bytes.

## 2.49.6 ARIA-CBC with PKCS padding

6777 ARIA-CBC with PKCS padding, denoted **CKM\_ARIA\_CBC\_PAD**, is a mechanism for single- and  
 6778 multiple-part encryption and decryption; key wrapping; and key unwrapping, based on ARIA; cipher-block  
 6779 chaining mode; and the block cipher padding method detailed in PKCS #7.

6780 It has a parameter, a 16-byte initialization vector.

6781 The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the  
 6782 ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified  
 6783 for the **CKA\_VALUE\_LEN** attribute.

6784 In addition to being able to wrap and unwrap secret keys, this mechanism can wrap and unwrap RSA,  
 6785 Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see Section  
 6786 TBA for details). The entries in the table below for data length constraints when wrapping and  
 6787 unwrapping keys do not apply to wrapping and unwrapping private keys.

6788 Constraints on key types and the length of data are summarized in the following table:

6789 *Table 187, ARIA-CBC with PKCS Padding: Key and Data Length*

| Function    | Key type | Input length           | Output length                                              |
|-------------|----------|------------------------|------------------------------------------------------------|
| C_Encrypt   | CKK_ARIA | any                    | input length rounded up to multiple of the block size      |
| C_Decrypt   | CKK_ARIA | multiple of block size | between 1 and block size bytes shorter than input length   |
| C_WrapKey   | CKK_ARIA | any                    | input length rounded up to multiple of the block size      |
| C_UnwrapKey | CKK_ARIA | multiple of block size | between 1 and block length bytes shorter than input length |

6790 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 6791 specify the supported range of ARIA key sizes, in bytes.

## 2.49.7 General-length ARIA-MAC

6793 General-length ARIA -MAC, denoted **CKM\_ARIA\_MAC\_GENERAL**, is a mechanism for single- and  
 6794 multiple-part signatures and verification, based on ARIA and data authentication as defined in [FIPS 113].

6795 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS** structure, which specifies the output length  
 6796 desired from the mechanism.

6797 The output bytes from this mechanism are taken from the start of the final ARIA cipher block produced in  
6798 the MACing process.

6799 Constraints on key types and the length of data are summarized in the following table:

6800 *Table 188, General-length ARIA-MAC: Key and Data Length*

| Function | Key type | Data length | Signature length                         |
|----------|----------|-------------|------------------------------------------|
| C_Sign   | CKK_ARIA | any         | 1-block size, as specified in parameters |
| C_Verify | CKK_ARIA | any         | 1-block size, as specified in parameters |

6801 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
6802 specify the supported range of ARIA key sizes, in bytes.

## 6803 **2.49.8 ARIA-MAC**

6804 ARIA-MAC, denoted by **CKM\_ARIA\_MAC**, is a special case of the general-length ARIA-MAC  
6805 mechanism. ARIA-MAC always produces and verifies MACs that are half the block size in length.

6806 It does not have a parameter.

6807 Constraints on key types and the length of data are summarized in the following table:

6808 *Table 189, ARIA-MAC: Key and Data Length*

| Function | Key type | Data length | Signature length                   |
|----------|----------|-------------|------------------------------------|
| C_Sign   | CKK_ARIA | any         | $\frac{1}{2}$ block size (8 bytes) |
| C_Verify | CKK_ARIA | any         | $\frac{1}{2}$ block size (8 bytes) |

6809 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
6810 specify the supported range of ARIA key sizes, in bytes.

## 6811 **2.50 Key derivation by data encryption - ARIA**

6812 These mechanisms allow derivation of keys using the result of an encryption operation as the key value.  
6813 They are for use with the C\_DeriveKey function.

### 6814 **2.50.1 Definitions**

6815 Mechanisms:

```
6816 CKM_ARIA_ECB_ENCRYPT_DATA
6817 CKM_ARIA_CBC_ENCRYPT_DATA
6818
6819 typedef struct CK_ARIA_CBC_ENCRYPT_DATA_PARAMS {
6820 CK_BYTE iv[16];
6821 CK_BYTE_PTR pData;
6822 CK ULONG length;
6823 } CK_ARIA_CBC_ENCRYPT_DATA_PARAMS;
6824
6825 typedef CK_ARIA_CBC_ENCRYPT_DATA_PARAMS CK_PTR
6826 CK_ARIA_CBC_ENCRYPT_DATA_PARAMS_PTR;
```

### 6827 **2.50.2 Mechanism Parameters**

6828 Uses CK\_ARIA\_CBC\_ENCRYPT\_DATA\_PARAMS, and CK\_KEY\_DERIVATION\_STRING\_DATA.

6829 *Table 190, Mechanism Parameters for Aria-based key derivation*

|                           |                                                                                                                                                       |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKM_ARIA_ECB_ENCRYPT_DATA | Uses CK_KEY_DERIVATION_STRING_DATA structure. Parameter is the data to be encrypted and must be a multiple of 16 long.                                |
| CKM_ARIA_CBC_ENCRYPT_DATA | Uses CK_ARIA_CBC_ENCRYPT_DATA_PARAMS. Parameter is an 16 byte IV value followed by the data. The data value part must be a multiple of 16 bytes long. |

6830

## 6831 **2.51 SEED**

6832 SEED is a symmetric block cipher developed by the South Korean Information Security Agency (KISA). It  
6833 has a 128-bit key size and a 128-bit block size.

6834 Its specification has been published as Internet [RFC 4269].

6835 RFCs have been published defining the use of SEED in

6836 TLS <ftp://ftp.rfc-editor.org/in-notes/rfc4162.txt>

6837 IPsec <ftp://ftp.rfc-editor.org/in-notes/rfc4196.txt>

6838 CMS <ftp://ftp.rfc-editor.org/in-notes/rfc4010.txt>

6839

6840 TLS cipher suites that use SEED include:

```
6841 CipherSuite TLS_RSA_WITH_SEED_CBC_SHA = { 0x00,
6842 0x96 };
6843 CipherSuite TLS_DH_DSS_WITH_SEED_CBC_SHA = { 0x00,
6844 0x97 };
6845 CipherSuite TLS_DH_RSA_WITH_SEED_CBC_SHA = { 0x00,
6846 0x98 };
6847 CipherSuite TLS_DHE_DSS_WITH_SEED_CBC_SHA = { 0x00,
6848 0x99 };
6849 CipherSuite TLS_DHE_RSA_WITH_SEED_CBC_SHA = { 0x00,
6850 0x9A };
6851 CipherSuite TLS_DH_anon_WITH_SEED_CBC_SHA = { 0x00,
6852 0x9B };
```

6853

6854 As with any block cipher, it can be used in the ECB, CBC, OFB and CFB modes of operation, as well as  
6855 in a MAC algorithm such as HMAC.

6856 OIDs have been published for all these uses. A list may be seen at

6857 <http://www.alvestrand.no/objectid/1.2.410.200004.1.html>

6858

6859 *Table 191, SEED Mechanisms vs. Functions*

| Mechanism                 | Functions         |               |                      |        |                    |               |        |
|---------------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                           | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SEED_KEY_GEN          |                   |               |                      |        | ✓                  |               |        |
| CKM_SEED_ECB              |                   |               | ✓                    |        |                    |               |        |
| CKM_SEED_CBC              |                   |               | ✓                    |        |                    |               |        |
| CKM_SEED_CBC_PAD          | ✓                 |               |                      |        |                    | ✓             |        |
| CKM_SEED_MAC_GENERAL      |                   |               | ✓                    |        |                    |               |        |
| CKM_SEED_MAC              |                   |               |                      | ✓      |                    |               |        |
| CKM_SEED_ECB_ENCRYPT_DATA |                   |               |                      |        |                    |               | ✓      |
| CKM_SEED_CBC_ENCRYPT_DATA |                   |               |                      |        |                    |               | ✓      |

## 2.51.1 Definitions

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

Mechanisms:

- CKM\_SEED\_KEY\_GEN
- CKM\_SEED\_ECB
- CKM\_SEED\_CBC
- CKM\_SEED\_MAC
- CKM\_SEED\_MAC\_GENERAL
- CKM\_SEED\_CBC\_PAD

For all of these mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** are always 16.

## 2.51.2 SEED secret key objects

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

Table 192, SEED Secret Key Object Attributes

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

- Refer to [PKCS11-Base] table 11 for footnotes.

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

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_SEED;
CK_UTF8CHAR label[] = "A SEED secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
```

```
6886 { CKA_CLASS, &class, sizeof(class) },
6887 { CKA_KEY_TYPE, &keyType, sizeof(keyType) },
6888 { CKA_TOKEN, &true, sizeof(true) },
6889 { CKA_LABEL, label, sizeof(label)-1 },
6890 { CKA_ENCRYPT, &true, sizeof(true) },
6891 { CKA_VALUE, value, sizeof(value) }
6892 };
```

### 6893 **2.51.3 SEED key generation**

6894 The SEED key generation mechanism, denoted CKM\_SEED\_KEY\_GEN, is a key generation mechanism  
6895 for SEED.

6896 It does not have a parameter.

6897 The mechanism generates SEED keys.

6898 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
6899 key. Other attributes supported by the SEED key type (specifically, the flags indicating which functions  
6900 the key supports) may be specified in the template for the key, or else are assigned default initial values.

### 6901 **2.51.4 SEED-ECB**

6902 SEED-ECB, denoted **CKM\_SEED\_ECB**, is a mechanism for single- and multiple-part encryption and  
6903 decryption; key wrapping; and key unwrapping, based on SEED and electronic codebook mode.

6904 It does not have a parameter.

### 6905 **2.51.5 SEED-CBC**

6906 SEED-CBC, denoted **CKM\_SEED\_CBC**, is a mechanism for single- and multiple-part encryption and  
6907 decryption; key wrapping; and key unwrapping, based on SEED and cipher-block chaining mode.

6908 It has a parameter, a 16-byte initialization vector.

### 6909 **2.51.6 SEED-CBC with PKCS padding**

6910 SEED-CBC with PKCS padding, denoted **CKM\_SEED\_CBC\_PAD**, is a mechanism for single- and  
6911 multiple-part encryption and decryption; key wrapping; and key unwrapping, based on SEED; cipher-  
6912 block chaining mode; and the block cipher padding method detailed in PKCS #7.

6913 It has a parameter, a 16-byte initialization vector.

### 6914 **2.51.7 General-length SEED-MAC**

6915 General-length SEED-MAC, denoted **CKM\_SEED\_MAC\_GENERAL**, is a mechanism for single- and  
6916 multiple-part signatures and verification, based on SEED and data authentication as defined in 0.

6917 It has a parameter, a **CK\_MAC\_GENERAL\_PARAMS** structure, which specifies the output length  
6918 desired from the mechanism.

6919 The output bytes from this mechanism are taken from the start of the final cipher block produced in the  
6920 MACing process.

### 6921 **2.51.8 SEED-MAC**

6922 SEED-MAC, denoted by **CKM\_SEED\_MAC**, is a special case of the general-length SEED-MAC  
6923 mechanism. SEED-MAC always produces and verifies MACs that are half the block size in length.

6924 It does not have a parameter.

6925 **2.52 Key derivation by data encryption - SEED**

6926 These mechanisms allow derivation of keys using the result of an encryption operation as the key value.  
6927 They are for use with the C\_DeriveKey function.

6928 **2.52.1 Definitions**

6929 Mechanisms:

```
6930 CKM_SEED_ECB_ENCRYPT_DATA
6931 CKM_SEED_CBC_ENCRYPT_DATA
6932
6933 typedef struct CK_SEED_CBC_ENCRYPT_DATA_PARAMS {
6934 CK_BYTE iv[16];
6935 CK_BYTE_PTR pData;
6936 CK_ULONG length;
6937 } CK_SEED_CBC_ENCRYPT_DATA_PARAMS;
6938
6939 typedef CK_SEED_CBC_ENCRYPT_DATA_PARAMS CK_PTR
6940 CK_SEED_CBC_ENCRYPT_DATA_PARAMS_PTR;
```

6941 **2.52.2 Mechanism Parameters**

6942 *Table 193, Mechanism Parameters for SEED-based key derivation*

|                           |                                                                                                                                                       |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKM_SEED_ECB_ENCRYPT_DATA | Uses CK_KEY_DERIVATION_STRING_DATA structure. Parameter is the data to be encrypted and must be a multiple of 16 long.                                |
| CKM_SEED_CBC_ENCRYPT_DATA | Uses CK_SEED_CBC_ENCRYPT_DATA_PARAMS. Parameter is an 16 byte IV value followed by the data. The data value part must be a multiple of 16 bytes long. |

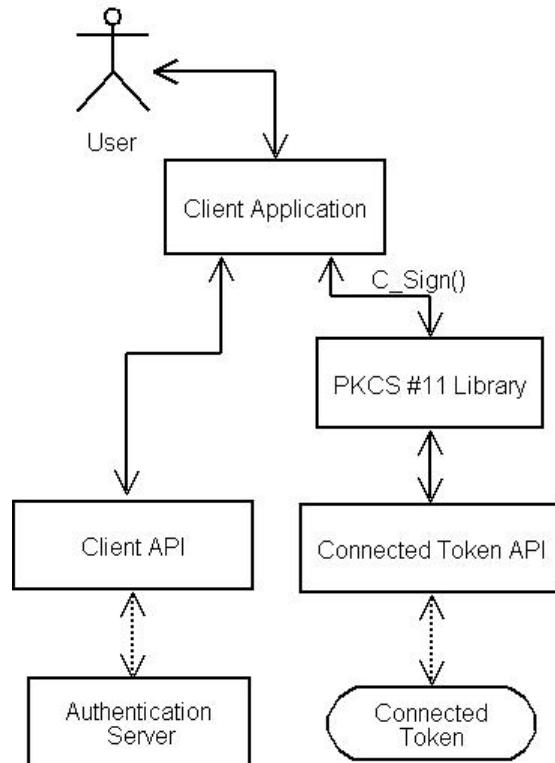
6943

6944 **2.53 OTP**

6945 **2.53.1 Usage overview**

6946 OTP tokens represented as PKCS #11 mechanisms may be used in a variety of ways. The usage cases  
6947 can be categorized according to the type of sought functionality.

6948    **2.53.2 Case 1: Generation of OTP values**

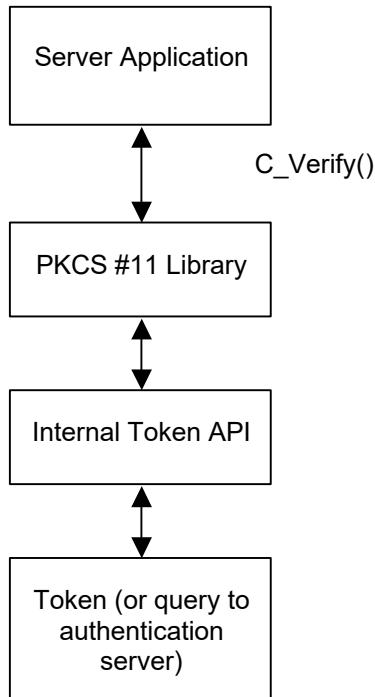


6949

6950    *Figure 1: Retrieving OTP values through C\_Sign*

6951    Figure 1 shows an integration of PKCS #11 into an application that needs to authenticate users holding  
6952    OTP tokens. In this particular example, a connected hardware token is used, but a software token is  
6953    equally possible. The application invokes **C\_Sign** to retrieve the OTP value from the token. In the  
6954    example, the application then passes the retrieved OTP value to a client API that sends it via the network  
6955    to an authentication server. The client API may implement a standard authentication protocol such as  
6956    RADIUS [RFC 2865] or EAP [RFC 3748], or a proprietary protocol such as that used by RSA Security's  
6957    ACE/Agent® software.

6958 **2.53.3 Case 2: Verification of provided OTP values**

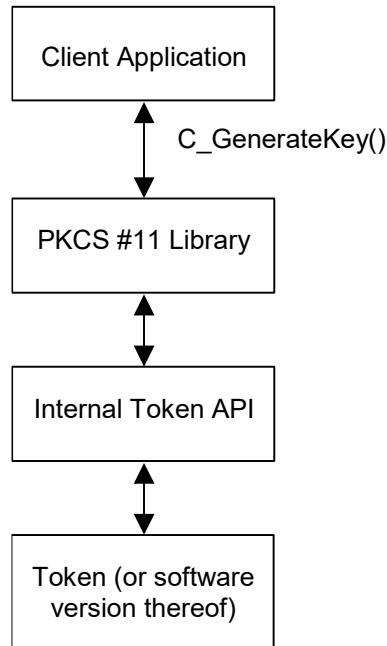


6959

6960 *Figure 2: Server-side verification of OTP values*

6961 Figure 2 illustrates the server-side equivalent of the scenario depicted in Figure 1. In this case, a server  
6962 application invokes **C\_Verify** with the received OTP value as the signature value to be verified.

6963 **2.53.4 Case 3: Generation of OTP keys**



6964

6965 *Figure 3: Generation of an OTP key*

6966 Figure 3 shows an integration of PKCS #11 into an application that generates OTP keys. The application  
6967 invokes **C\_GenerateKey** to generate an OTP key of a particular type on the token. The key may  
6968 subsequently be used as a basis to generate OTP values.

## 6969 **2.53.5 OTP objects**

### 6970 **2.53.5.1 Key objects**

6971 OTP key objects (object class **CKO OTP KEY**) hold secret keys used by OTP tokens. The following  
6972 table defines the attributes common to all OTP keys, in addition to the attributes defined for secret keys,  
6973 all of which are inherited by this class:

6974 Table 194: Common OTP key attributes

| Attribute                                  | Data type | Meaning                                                                                                                                                                                                                                                                                 |
|--------------------------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKA_OTP_FORMAT                             | CK ULONG  | Format of OTP values produced with this key:<br>CK_OTP_FORMAT_DECIMAL = Decimal (default) (UTF8-encoded)<br>CK_OTP_FORMAT_HEXADECIMAL = Hexadecimal (UTF8-encoded)<br>CK_OTP_FORMAT_ALPHANUMERIC = Alphanumeric (UTF8-encoded)<br>CK_OTP_FORMAT_BINARY = Only binary values.            |
| CKA_OTP_LENGTH <sup>9</sup>                | CK ULONG  | Default length of OTP values (in the CKA_OTP_FORMAT) produced with this key.                                                                                                                                                                                                            |
| CKA_OTP_USER_FRIENDLY_MODE <sup>9</sup>    | CK_BBOOL  | Set to CK_TRUE when the token is capable of returning OTPs suitable for human consumption. See the description of CKF_USER_FRIENDLY OTP below.                                                                                                                                          |
| CKA_OTP_CHALLENGE_REQUIREMENT <sup>9</sup> | CK ULONG  | Parameter requirements when generating or verifying OTP values with this key:<br>CK_OTP_PARAM_MANDATORY = A challenge must be supplied.<br>CK_OTP_PARAM_OPTIONAL = A challenge may be supplied but need not be.<br>CK_OTP_PARAM_IGNORED = A challenge, if supplied, will be ignored.    |
| CKA_OTP_TIME_REQUIREMENT <sup>9</sup>      | CK ULONG  | Parameter requirements when generating or verifying OTP values with this key:<br>CK_OTP_PARAM_MANDATORY = A time value must be supplied.<br>CK_OTP_PARAM_OPTIONAL = A time value may be supplied but need not be.<br>CK_OTP_PARAM_IGNORED = A time value, if supplied, will be ignored. |

|                                          |                 |                                                                                                                                                                                                                                                                                                                                                           |
|------------------------------------------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKA OTP COUNTER REQUIREMENT <sup>9</sup> | CK ULONG        | Parameter requirements when generating or verifying OTP values with this key:<br>CK OTP PARAM MANDATORY = A counter value must be supplied.<br>CK OTP PARAM OPTIONAL = A counter value may be supplied but need not be.<br>CK OTP PARAM IGNORED = A counter value, if supplied, will be ignored.                                                          |
| CKA OTP PIN REQUIREMENT <sup>9</sup>     | CK ULONG        | Parameter requirements when generating or verifying OTP values with this key:<br>CK OTP PARAM MANDATORY = A PIN value must be supplied.<br>CK OTP PARAM OPTIONAL = A PIN value may be supplied but need not be (if not supplied, then library will be responsible for collecting it)<br>CK OTP PARAM IGNORED = A PIN value, if supplied, will be ignored. |
| CKA OTP COUNTER                          | Byte array      | Value of the associated internal counter. Default value is empty (i.e. ulValueLen = 0).                                                                                                                                                                                                                                                                   |
| CKA OTP TIME                             | RFC 2279 string | Value of the associated internal UTC time in the form YYYYMMDDhhmmss. Default value is empty (i.e. ulValueLen= 0).                                                                                                                                                                                                                                        |
| CKA OTP USER IDENTIFIER                  | RFC 2279 string | Text string that identifies a user associated with the OTP key (may be used to enhance the user experience). Default value is empty (i.e. ulValueLen = 0).                                                                                                                                                                                                |
| CKA OTP SERVICE IDENTIFIER               | RFC 2279 string | Text string that identifies a service that may validate OTPs generated by this key. Default value is empty (i.e. ulValueLen = 0).                                                                                                                                                                                                                         |
| CKA OTP SERVICE LOGO                     | Byte array      | Logotype image that identifies a service that may validate OTPs generated by this key. Default value is empty (i.e. ulValueLen = 0).                                                                                                                                                                                                                      |
| CKA OTP SERVICE LOGO TYPE                | RFC 2279 string | MIME type of the CKA OTP SERVICE LOGO attribute value. Default value is empty (i.e. ulValueLen = 0).                                                                                                                                                                                                                                                      |
| CKA VALUE <sup>1, 4, 6, 7</sup>          | Byte array      | Value of the key.                                                                                                                                                                                                                                                                                                                                         |
| CKA VALUE LEN <sup>2, 3</sup>            | CK ULONG        | Length in bytes of key value.                                                                                                                                                                                                                                                                                                                             |

6976 Note: A Cryptoki library may support PIN-code caching in order to reduce user interactions. An OTP-  
6977 PKCS #11 application should therefore always consult the state of the CKA OTP\_PIN\_REQUIREMENT  
6978 attribute before each call to **C\_SignInit**, as the value of this attribute may change dynamically.

6979 For OTP tokens with multiple keys, the keys may be enumerated using **C\_FindObjects**. The  
6980 **CKA OTP\_SERVICE\_IDENTIFIER** and/or the **CKA OTP\_SERVICE\_LOGO** attribute may be used to  
6981 distinguish between keys. The actual choice of key for a particular operation is however application-  
6982 specific and beyond the scope of this document.

6983 For all OTP keys, the CKA\_ALLOWED\_MECHANISMS attribute should be set as required.

## 6984 **2.53.6 OTP-related notifications**

6985 This document extends the set of defined notifications as follows:

6986       **CKN OTP CHANGED**     *Cryptoki is informing the application that the OTP for a key on a  
6987 connected token just changed. This notification is particularly useful  
6988 when applications wish to display the current OTP value for time-  
6989 based mechanisms.*

## 6990 **2.53.7 OTP mechanisms**

6991 The following table shows, for the OTP mechanisms defined in this document, their support by different  
6992 cryptographic operations. For any particular token, of course, a particular operation may well support  
6993 only a subset of the mechanisms listed. There is also no guarantee that a token that supports one  
6994 mechanism for some operation supports any other mechanism for any other operation (or even supports  
6995 that same mechanism for any other operation).

6996 *Table 195: OTP mechanisms vs. applicable functions*

| Mechanism           | Functions         |               |                      |        |                    |               |        |
|---------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                     | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SECURID_KEY_GEN |                   |               |                      |        | ✓                  |               |        |
| CKM_SECURID         |                   | ✓             |                      |        |                    |               |        |
| CKM_HOTP_KEY_GEN    |                   |               |                      |        | ✓                  |               |        |
| CKM_HOTP            |                   | ✓             |                      |        |                    |               |        |
| CKM_ACTI_KEY_GEN    |                   |               |                      |        | ✓                  |               |        |
| CKM_ACTI            |                   | ✓             |                      |        |                    |               |        |

6997 The remainder of this section will present in detail the OTP mechanisms and the parameters that are  
6998 supplied to them.

### 6999 **2.53.7.1 OTP mechanism parameters**

#### 7000 ◆ **CK OTP PARAM TYPE**

7001 **CK OTP PARAM TYPE** is a value that identifies an OTP parameter type. It is defined as follows:

7002       **typedef CK ULONG CK OTP PARAM TYPE;**

7003 The following **CK OTP PARAM TYPE** types are defined:

7004 *Table 196, OTP parameter types*

| <b>Parameter</b>     | <b>Data type</b> | <b>Meaning</b>                                                                                                                                                                                                                                                                            |
|----------------------|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CK OTP PIN           | RFC 2279 string  | A UTF8 string containing a PIN for use when computing or verifying PIN-based OTP values.                                                                                                                                                                                                  |
| CK OTP CHALLENGE     | Byte array       | Challenge to use when computing or verifying challenge-based OTP values.                                                                                                                                                                                                                  |
| CK OTP TIME          | RFC 2279 string  | UTC time value in the form YYYYMMDDhhmmss to use when computing or verifying time-based OTP values.                                                                                                                                                                                       |
| CK OTP COUNTER       | Byte array       | Counter value to use when computing or verifying counter-based OTP values.                                                                                                                                                                                                                |
| CK OTP FLAGS         | CK_FLAGS         | Bit flags indicating the characteristics of the sought OTP as defined below.                                                                                                                                                                                                              |
| CK OTP OUTPUT LENGTH | CK ULONG         | Desired output length (overrides any default value). A Cryptoki library will return CKR_MECHANISM_PARAM_INVALID if a provided length value is not supported.                                                                                                                              |
| CK OTP OUTPUT FORMAT | CK ULONG         | Returned OTP format (allowed values are the same as for CKA OTP FORMAT). This parameter is only intended for <b>C_Sign</b> output, see paragraphs below. When not present, the returned OTP format will be the same as the value of the CKA OTP FORMAT attribute for the key in question. |
| CK OTP VALUE         | Byte array       | An actual OTP value. This parameter type is intended for <b>C_Sign</b> output, see paragraphs below.                                                                                                                                                                                      |

7005

7006 The following table defines the possible values for the CK OTP FLAGS type:

7007 *Table 197: OTP Mechanism Flags*

| <b>Bit flag</b> | <b>Mask</b> | <b>Meaning</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKF_NEXT OTP    | 0x00000001  | True (i.e. set) if the OTP computation shall be for the next OTP, rather than the current one (current being interpreted in the context of the algorithm, e.g. for the current counter value or current time window). A Cryptoki library shall return CKR_MECHANISM_PARAM_INVALID if the CKF_NEXT OTP flag is set and the OTP mechanism in question does not support the concept of “next” OTP or the library is not capable of generating the next OTP <sup>9</sup> . |

<sup>9</sup> Applications that may need to retrieve the next OTP should be prepared to handle this situation. For example, an application could store the OTP value returned by C\_Sign so that, if a next OTP is required, it can compare it to the OTP value returned by subsequent calls to C\_Sign should it turn out that the library does not support the CKF\_NEXT OTP flag.

| <b>Bit flag</b>       | <b>Mask</b> | <b>Meaning</b>                                                                                                                                                                                                                                                                                                                                                        |
|-----------------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKF_EXCLUDE_TIME      | 0x00000002  | True (i.e. set) if the OTP computation must not include a time value. Will have an effect only on mechanisms that do include a time value in the OTP computation and then only if the mechanism (and token) allows exclusion of this value. A Cryptoki library shall return CKR_MECHANISM_PARAM_INVALID if exclusion of the value is not allowed.                     |
| CKF_EXCLUDE_COUNTER   | 0x00000004  | True (i.e. set) if the OTP computation must not include a counter value. Will have an effect only on mechanisms that do include a counter value in the OTP computation and then only if the mechanism (and token) allows exclusion of this value. A Cryptoki library shall return CKR_MECHANISM_PARAM_INVALID if exclusion of the value is not allowed.               |
| CKF_EXCLUDE_CHALLENGE | 0x00000008  | True (i.e. set) if the OTP computation must not include a challenge. Will have an effect only on mechanisms that do include a challenge in the OTP computation and then only if the mechanism (and token) allows exclusion of this value. A Cryptoki library shall return CKR_MECHANISM_PARAM_INVALID if exclusion of the value is not allowed.                       |
| CKF_EXCLUDE_PIN       | 0x00000010  | True (i.e. set) if the OTP computation must not include a PIN value. Will have an effect only on mechanisms that do include a PIN in the OTP computation and then only if the mechanism (and token) allows exclusion of this value. A Cryptoki library shall return CKR_MECHANISM_PARAM_INVALID if exclusion of the value is not allowed.                             |
| CKF_USER_FRIENDLY OTP | 0x00000020  | True (i.e. set) if the OTP returned shall be in a form suitable for human consumption. If this flag is set, and the call is successful, then the returned CK_OTP_VALUE shall be a UTF8-encoded printable string. A Cryptoki library shall return CKR_MECHANISM_PARAM_INVALID if this flag is set when CKA_OTP_USER_FRIENDLY_MODE for the key in question is CK_FALSE. |

7008 Note: Even if CKA\_OTP\_FORMAT is not set to CK\_OTP\_FORMAT\_BINARY, then there may still be  
 7009 value in setting the CKF\_USER\_FRIENDLY OTP flag (assuming CKA\_OTP\_USER\_FRIENDLY\_MODE  
 7010 is CK\_TRUE, of course) if the intent is for a human to read the generated OTP value, since it may  
 7011 become shorter or otherwise better suited for a user. Applications that do not intend to provide a returned  
 7012 OTP value to a user should not set the CKF\_USER\_FRIENDLY OTP flag.

#### 7013 ◆ **CK\_OTP\_PARAM; CK\_OTP\_PARAM\_PTR**

7014 **CK\_OTP\_PARAM** is a structure that includes the type, value, and length of an OTP parameter. It is  
 7015 defined as follows:

```

7016 typedef struct CK OTP PARAM {
7017 CK OTP PARAM TYPE type;
7018 CK VOID PTR pValue;
7019 CK ULONG ulValueLen;
7020 } CK OTP PARAM;
7021
7022 The fields of the structure have the following meanings:
7023 type the parameter type
7024 pValue pointer to the value of the parameter
7025 ulValueLen length in bytes of the value
7026
7027 If a parameter has no value, then ulValueLen = 0, and the value of pValue is irrelevant. Note that pValue is a “void” pointer, facilitating the passing of arbitrary values. Both the application and the Cryptoki library must ensure that the pointer can be safely cast to the expected type (i.e., without word-alignment errors).
7028 CK OTP PARAM PTR is a pointer to a CK OTP PARAM.
7029
7030 ♦ CK OTP PARAMS; CK OTP PARAMS PTR
7031 CK OTP PARAMS is a structure that is used to provide parameters for OTP mechanisms in a generic fashion. It is defined as follows:
7032
7033 typedef struct CK OTP PARAMS {
7034 CK OTP PARAM PTR pParams;
7035 CK ULONG ulCount;
7036 } CK OTP PARAMS;
7037
7038 The fields of the structure have the following meanings:
7039 pParams pointer to an array of OTP parameters
7040 ulCount the number of parameters in the array
7041
7042 CK OTP PARAMS PTR is a pointer to a CK OTP PARAMS.
7043
7044 When calling C_SignInit or C_VerifyInit with a mechanism that takes a CK OTP PARAMS structure as a parameter, the CK OTP PARAMS structure shall be populated in accordance with the CKA OTP X REQUIREMENT key attributes for the identified key, where X is PIN, CHALLENGE, TIME, or COUNTER.
7045
7046 For example, if CKA OTP TIME REQUIREMENT = CK OTP PARAM MANDATORY, then the CK OTP TIME parameter shall be present. If CKA OTP TIME REQUIREMENT = CK OTP PARAM OPTIONAL, then a CK OTP TIME parameter may be present. If it is not present, then the library may collect it (during the C_Sign call). If CKA OTP TIME REQUIREMENT = CK OTP PARAM IGNORED, then a provided CK OTP TIME parameter will always be ignored.
7047 Additionally, a provided CK OTP TIME parameter will always be ignored if CKF EXCLUDE TIME is set in a CK OTP FLAGS parameter. Similarly, if this flag is set, a library will not attempt to collect the value itself, and it will also instruct the token not to make use of any internal value, subject to token policies. It is an error (CKR MECHANISM PARAM INVALID) to set the CKF EXCLUDE TIME flag when the CKA OTP TIME REQUIREMENT attribute is CK OTP PARAM MANDATORY.
7048
7049 The above discussion holds for all CKA OTP X REQUIREMENT attributes (i.e.,
7050 CKA OTP PIN REQUIREMENT, CKA OTP CHALLENGE REQUIREMENT,
7051 CKA OTP COUNTER REQUIREMENT, CKA OTP TIME REQUIREMENT). A library may set a particular CKA OTP X REQUIREMENT attribute to CK OTP PARAM OPTIONAL even if it is required

```

7060 by the mechanism as long as the token (or the library itself) has the capability of providing the value to the  
7061 computation. One example of this is a token with an on-board clock.

7062 In addition, applications may use the CK OTP FLAGS, the CK OTP OUTPUT FORMAT and the  
7063 CKA OTP LENGTH parameters to set additional parameters.

7064

7065 ◆ **CK OTP SIGNATURE INFO, CK OTP SIGNATURE INFO PTR**

7066 **CK OTP SIGNATURE INFO** is a structure that is returned by all OTP mechanisms in successful calls to  
7067 **C\_Sign** (**C\_SignFinal**). The structure informs applications of actual parameter values used in particular  
7068 OTP computations in addition to the OTP value itself. It is used by all mechanisms for which the key  
7069 belongs to the class **CKO OTP KEY** and is defined as follows:

```
7070 typedef struct CK OTP SIGNATURE INFO {
7071 CK OTP PARAM PTR pParams;
7072 CK ULONG ulCount;
7073 } CK OTP SIGNATURE INFO;
```

7074 The fields of the structure have the following meanings:

7075 *pParams* pointer to an array of OTP parameter values

7076 *ulCount* the number of parameters in the array

7077 After successful calls to **C\_Sign** or **C\_SignFinal** with an OTP mechanism, the *pSignature* parameter will  
7078 be set to point to a **CK OTP SIGNATURE INFO** structure. One of the parameters in this structure will be  
7079 the OTP value itself, identified with the **CK OTP VALUE** tag. Other parameters may be present for  
7080 informational purposes, e.g. the actual time used in the OTP calculation. In order to simplify OTP  
7081 validations, authentication protocols may permit authenticating parties to send some or all of these  
7082 parameters in addition to OTP values themselves. Applications should therefore check for their presence  
7083 in returned **CK OTP SIGNATURE INFO** values whenever such circumstances apply.

7084 Since **C\_Sign** and **C\_SignFinal** follows the convention described in [PKCS11-Base] Section 5.2 on  
7085 producing output, a call to **C\_Sign** (or **C\_SignFinal**) with *pSignature* set to **NULL\_PTR** will return (in the  
7086 *pulSignatureLen* parameter) the required number of bytes to hold the **CK OTP SIGNATURE INFO**  
7087 structure as well as all the data in all its **CK OTP PARAM** components. If an application allocates a  
7088 memory block based on this information, it shall therefore not subsequently de-allocate components of  
7089 such a received value but rather de-allocate the complete **CK OTP PARAMS** structure itself. A Cryptoki  
7090 library that is called with a non-**NULL** *pSignature* pointer will assume that it points to a *contiguous*  
7091 memory block of the size indicated by the *pulSignatureLen* parameter.

7092 When verifying an OTP value using an OTP mechanism, *pSignature* shall be set to the OTP value itself,  
7093 e.g. the value of the **CK OTP VALUE** component of a **CK OTP PARAM** structure returned by a call to  
7094 **C\_Sign**. The **CK OTP PARAM** value supplied in the **C\_VerifyInit** call sets the values to use in the  
7095 verification operation.

7096 **CK OTP SIGNATURE INFO PTR** points to a **CK OTP SIGNATURE INFO**.

7097 **2.53.8 RSA SecurID**

7098 **2.53.8.1 RSA SecurID secret key objects**

7099 RSA SecurID secret key objects (object class **CKO OTP KEY**, key type **CKK SECURID**) hold RSA  
7100 SecurID secret keys. The following table defines the RSA SecurID secret key object attributes, in  
7101 addition to the common attributes defined for this object class:

7102 Table 198, RSA SecurID secret key object attributes

| Attribute                          | Data type | Meaning                                                                        |
|------------------------------------|-----------|--------------------------------------------------------------------------------|
| CKA OTP TIME INTERVAL <sup>1</sup> | CK ULONG  | Interval between OTP values produced with this key, in seconds. Default is 60. |

7103 Refer to [PKCS11-Base] table 11 for footnotes.

7104 The following is a sample template for creating an RSA SecurID secret key object:

```

7105 CK_OBJECT_CLASS class = CKO OTP KEY;
7106 CK_KEY_TYPE keyType = CKK_SECURID;
7107 CK_DATE endDate = {...};
7108 CK_UTF8CHAR label[] = "RSA SecurID secret key object";
7109 CK_BYTE keyId[] = {...};
7110 CK ULONG outputFormat = CK OTP FORMAT DECIMAL;
7111 CK ULONG outputLength = 6;
7112 CK ULONG needPIN = CK OTP PARAM MANDATORY;
7113 CK ULONG timeInterval = 60;
7114 CK_BYTE value[] = {...};
7115 CK_BBOOL true = CK TRUE;
7116 CK_ATTRIBUTE template[] = {
7117 {CKA_CLASS, &class, sizeof(class)},
7118 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
7119 {CKA_END_DATE, &endDate, sizeof(endDate)},
7120 {CKA_TOKEN, &true, sizeof(true)},
7121 {CKA_SENSITIVE, &true, sizeof(true)},
7122 {CKA_LABEL, label, sizeof(label)-1},
7123 {CKA_SIGN, &true, sizeof(true)},
7124 {CKA_VERIFY, &true, sizeof(true)},
7125 {CKA_ID, keyId, sizeof(keyId)},
7126 {CKA OTP FORMAT, &outputFormat, sizeof(outputFormat)},
7127 {CKA OTP LENGTH, &outputLength, sizeof(outputLength)},
7128 {CKA OTP PIN REQUIREMENT, &needPIN, sizeof(needPIN)},
7129 {CKA OTP TIME INTERVAL, &timeInterval,
7130 sizeof(timeInterval)},
7131 {CKA_VALUE, value, sizeof(value)}}
7132 };

```

7133 

### 2.53.8.2 RSA SecurID key generation

7134 The RSA SecurID key generation mechanism, denoted **CKM\_SECURID\_KEY\_GEN**, is a key generation  
7135 mechanism for the RSA SecurID algorithm.

7136 It does not have a parameter.

7137 The mechanism generates RSA SecurID keys with a particular set of attributes as specified in the  
7138 template for the key.7139 The mechanism contributes at least the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_VALUE\_LEN**, and  
7140 **CKA\_VALUE** attributes to the new key. Other attributes supported by the RSA SecurID key type may be  
7141 specified in the template for the key, or else are assigned default initial values.7142 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7143 specify the supported range of SecurID key sizes, in bytes.

### 2.53.8.3 SecurID OTP generation and validation

**CKM\_SECURID** is the mechanism for the retrieval and verification of RSA SecurID OTP values.

The mechanism takes a pointer to a **CK OTP\_PARAMS** structure as a parameter.

When signing or verifying using the **CKM\_SECURID** mechanism, *pData* shall be set to **NULL\_PTR** and *ulDataLen* shall be set to 0.

### 2.53.8.4 Return values

Support for the **CKM\_SECURID** mechanism extends the set of return values for **C\_Verify** with the following values:

- **CKR\_NEW\_PIN\_MODE**: The supplied OTP was not accepted and the library requests a new OTP computed using a new PIN. The new PIN is set through means out of scope for this document.
- **CKR\_NEXT OTP**: The supplied OTP was correct but indicated a larger than normal drift in the token's internal state (e.g. clock, counter). To ensure this was not due to a temporary problem, the application should provide the next one-time password to the library for verification.

## 2.53.9 OATH HOTP

### 2.53.9.1 OATH HOTP secret key objects

HOTP secret key objects (object class **CKO OTP KEY**, key type **CKK\_HOTP**) hold generic secret keys and associated counter values.

The **CKA OTP COUNTER** value may be set at key generation; however, some tokens may set it to a fixed initial value. Depending on the token's security policy, this value may not be modified and/or may not be revealed if the object has its **CKA SENSITIVE** attribute set to **CK\_TRUE** or its **CKA\_EXTRACTABLE** attribute set to **CK\_FALSE**.

For HOTP keys, the **CKA OTP COUNTER** value shall be an 8 bytes unsigned integer in big endian (i.e. network byte order) form. The same holds true for a **CK OTP COUNTER** value in a **CK OTP PARAM** structure.

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

```
7169 CK_OBJECT_CLASS class = CKO OTP KEY;
7170 CK_KEY_TYPE keyType = CKK_HOTP;
7171 CK_UTF8CHAR label[] = "HOTP secret key object";
7172 CK_BYTE keyId[] = {...};
7173 CK_ULONG outputFormat = CK OTP FORMAT DECIMAL;
7174 CK_ULONG outputLength = 6;
7175 CK_DATE endDate = {...};
7176 CK_BYTE counterValue[8] = {0};
7177 CK_BYTE value[] = {...};
7178 CK_BBOOL true = CK TRUE;
7179 CK_ATTRIBUTE template[] = {
7180 {CKA CLASS, &class, sizeof(class)},
7181 {CKA KEY TYPE, &keyType, sizeof(keyType)},
7182 {CKA END DATE, &endDate, sizeof(endDate)},
7183 {CKA TOKEN, &true, sizeof(true)},
7184 {CKA SENSITIVE, &true, sizeof(true)},
7185 {CKA LABEL, label, sizeof(label)-1},
7186 {CKA SIGN, &true, sizeof(true)},
7187 {CKA VERIFY, &true, sizeof(true)},
7188 {CKA ID, keyId, sizeof(keyId)},
```

```
7189 { CKA OTP FORMAT, &outputFormat, sizeof(outputFormat) },
7190 { CKA OTP LENGTH, &outputLength, sizeof(outputLength) },
7191 { CKA OTP COUNTER, counterValue, sizeof(counterValue) },
7192 { CKA VALUE, value, sizeof(value) }
7193 };
```

## 7194 2.53.9.2 HOTP key generation

7195 The HOTP key generation mechanism, denoted **CKM\_HOTP\_KEY\_GEN**, is a key generation mechanism  
7196 for the HOTP algorithm.

7197 It does not have a parameter.

7198 The mechanism generates HOTP keys with a particular set of attributes as specified in the template for  
7199 the key.

7200 The mechanism contributes at least the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA OTP COUNTER**,  
7201 **CKA\_VALUE** and **CKA\_VALUE\_LEN** attributes to the new key. Other attributes supported by the HOTP  
7202 key type may be specified in the template for the key, or else are assigned default initial values.

7203 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7204 specify the supported range of HOTP key sizes, in bytes.

## 7205 2.53.9.3 HOTP OTP generation and validation

7206 **CKM\_HOTP** is the mechanism for the retrieval and verification of HOTP OTP values based on the current  
7207 internal counter, or a provided counter.

7208 The mechanism takes a pointer to a **CK OTP PARAMS** structure as a parameter.

7209 As for the **CKM\_SECURID** mechanism, when signing or verifying using the **CKM\_HOTP** mechanism,  
7210 *pData* shall be set to **NULL\_PTR** and *ulDataLen* shall be set to 0.

7211 For verify operations, the counter value **CK OTP COUNTER** must be provided as a **CK OTP PARAM**  
7212 parameter to **C VerifyInit**. When verifying an OTP value using the **CKM\_HOTP** mechanism, *pSignature*  
7213 shall be set to the OTP value itself, e.g. the value of the **CK OTP VALUE** component of a  
7214 **CK OTP PARAM** structure in the case of an earlier call to **C\_Sign**.

## 7215 2.53.10 ActivIdentity ACTI

### 7216 2.53.10.1 ACTI secret key objects

7217 ACTI secret key objects (object class **CKO OTP KEY**, key type **CKK\_ACTI**) hold ActivIdentity ACTI  
7218 secret keys.

7219 For ACTI keys, the **CKA OTP COUNTER** value shall be an 8 bytes unsigned integer in big endian (i.e.  
7220 network byte order) form. The same holds true for the **CK OTP COUNTER** value in the  
7221 **CK OTP PARAM** structure.

7222 The **CKA OTP COUNTER** value may be set at key generation; however, some tokens may set it to a  
7223 fixed initial value. Depending on the token's security policy, this value may not be modified and/or may  
7224 not be revealed if the object has its **CKA SENSITIVE** attribute set to CK\_TRUE or its  
7225 **CKA EXTRACTABLE** attribute set to CK\_FALSE.

7226 The **CKA OTP TIME** value may be set at key generation; however, some tokens may set it to a fixed  
7227 initial value. Depending on the token's security policy, this value may not be modified and/or may not be  
7228 revealed if the object has its **CKA SENSITIVE** attribute set to CK\_TRUE or its **CKA EXTRACTABLE**  
7229 attribute set to CK\_FALSE.

7230 The following is a sample template for creating an ACTI secret key object:

```
7231 CK_OBJECT_CLASS class = CKO OTP KEY;
7232 CK_KEY_TYPE keyType = CKK_ACTI;
7233 CK_UTF8CHAR label[] = "ACTI secret key object";
```

```

7234 CK_BYTE keyId[] = { ... };
7235 CK_ULONG outputFormat = CK OTP FORMAT DECIMAL;
7236 CK_ULONG outputLength = 6;
7237 CK_DATE endDate = { ... };
7238 CK_BYTE counterValue[8] = { 0 };
7239 CK_BYTE value[] = { ... };
7240 CK_BBOOL true = CK_TRUE;
7241 CK_ATTRIBUTE template[] = {
7242 { CKA_CLASS, &class, sizeof(class) },
7243 { CKA_KEY_TYPE, &keyType, sizeof(keyType) },
7244 { CKA_END_DATE, &endDate, sizeof(endDate) },
7245 { CKA_TOKEN, &true, sizeof(true) },
7246 { CKA_SENSITIVE, &true, sizeof(true) },
7247 { CKA_LABEL, label, sizeof(label)-1 },
7248 { CKA_SIGN, &true, sizeof(true) },
7249 { CKA_VERIFY, &true, sizeof(true) },
7250 { CKA_ID, keyId, sizeof(keyId) },
7251 { CKA OTP FORMAT, &outputFormat,
7252 sizeof(outputFormat) },
7253 { CKA OTP LENGTH, &outputLength,
7254 sizeof(outputLength) },
7255 { CKA OTP COUNTER, counterValue,
7256 sizeof(counterValue) },
7257 { CKA VALUE, value, sizeof(value) }
7258 };

```

### 2.53.10.2 ACTI key generation

7260 The ACTI key generation mechanism, denoted **CKM\_ACTI\_KEY\_GEN**, is a key generation mechanism  
 7261 for the ACTI algorithm.

7262 It does not have a parameter.

7263 The mechanism generates ACTI keys with a particular set of attributes as specified in the template for the  
 7264 key.

7265 The mechanism contributes at least the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_VALUE** and  
 7266 **CKA\_VALUE\_LEN** attributes to the new key. Other attributes supported by the ACTI key type may be  
 7267 specified in the template for the key, or else are assigned default initial values.

7268 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 7269 specify the supported range of ACTI key sizes, in bytes.

### 2.53.10.3 ACTI OTP generation and validation

7271 **CKM\_ACTI** is the mechanism for the retrieval and verification of ACTI OTP values.

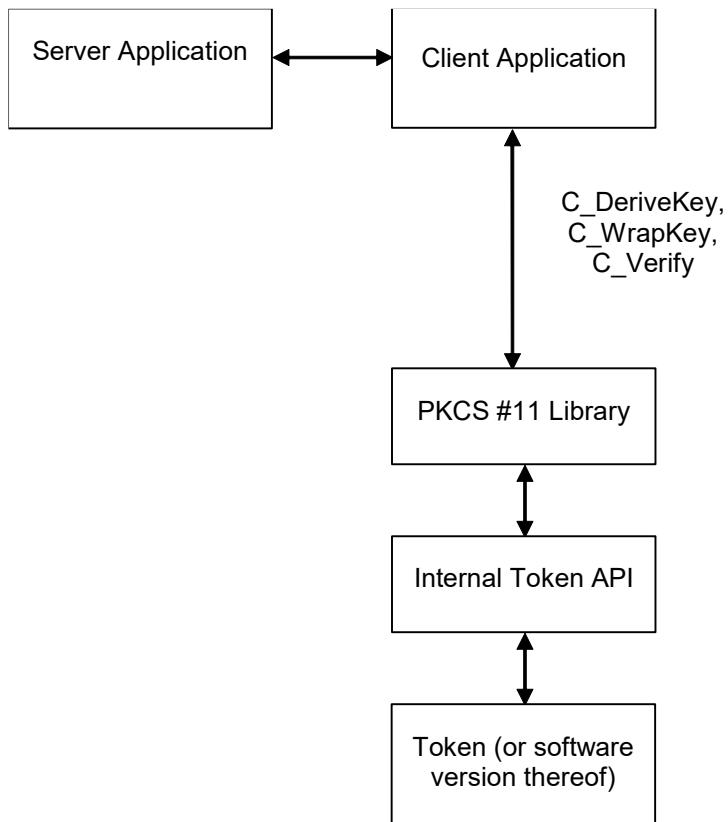
7272 The mechanism takes a pointer to a **CK OTP PARAMS** structure as a parameter.

7273 When signing or verifying using the **CKM\_ACTI** mechanism, *pData* shall be set to **NULL\_PTR** and  
 7274 *ulDataLen* shall be set to 0.

7275 When verifying an OTP value using the **CKM\_ACTI** mechanism, *pSignature* shall be set to the OTP value  
 7276 itself, e.g. the value of the **CK OTP VALUE** component of a **CK OTP PARAM** structure in the case of  
 7277 an earlier call to **C\_Sign**.

7278 **2.54 CT-KIP**

7279 **2.54.1 Principles of Operation**



7280

7281 *Figure 4: PKCS #11 and CT-KIP integration*

7282 Figure 4 shows an integration of PKCS #11 into an application that generates cryptographic keys through  
7283 the use of CT-KIP. The application invokes **C\_DeriveKey** to derive a key of a particular type on the token.  
7284 The key may subsequently be used as a basis to e.g., generate one-time password values. The  
7285 application communicates with a CT-KIP server that participates in the key derivation and stores a copy  
7286 of the key in its database. The key is transferred to the server in wrapped form, after a call to  
7287 **C\_WrapKey**. The server authenticates itself to the client and the client verifies the authentication by calls  
7288 to **C\_Verify**.

7289 **2.54.2 Mechanisms**

7290 The following table shows, for the mechanisms defined in this document, their support by different  
7291 cryptographic operations. For any particular token, of course, a particular operation may well support  
7292 only a subset of the mechanisms listed. There is also no guarantee that a token that supports one  
7293 mechanism for some operation supports any other mechanism for any other operation (or even supports  
7294 that same mechanism for any other operation).

7295 Table 199: CT-KIP Mechanisms vs. applicable functions

| Mechanism      | Functions         |               |                      |        |                    |               |        |
|----------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_KIP_DERIVE |                   |               |                      |        |                    |               | ✓      |
| CKM_KIP_WRAP   |                   |               |                      |        |                    | ✓             |        |
| CKM_KIP_MAC    |                   | ✓             |                      |        |                    |               |        |

7296 The remainder of this section will present in detail the mechanisms and the parameters that are supplied  
7297 to them.7298 

### 2.54.3 Definitions

7299 Mechanisms:

7300 CKM\_KIP\_DERIVE

7301 CKM\_KIP\_WRAP

7302 CKM\_KIP\_MAC

7303 

### 2.54.4 CT-KIP Mechanism parameters

7304 ◆ **CK\_KIP\_PARAMS; CK\_KIP\_PARAMS\_PTR**7305 **CK\_KIP\_PARAMS** is a structure that provides the parameters to all the CT-KIP related mechanisms: The  
7306 **CKM\_KIP\_DERIVE** key derivation mechanism, the **CKM\_KIP\_WRAP** key wrap and key unwrap  
7307 mechanism, and the **CKM\_KIP\_MAC** signature mechanism. The structure is defined as follows:

```
7308 typedef struct CK_KIP_PARAMS {
7309 CK_MECHANISM_PTR pMechanism;
7310 CK_OBJECT_HANDLE hKey;
7311 CK_BYTE_PTR pSeed;
7312 CK ULONG ulSeedLen;
7313 } CK_KIP_PARAMS;
```

7314 The fields of the structure have the following meanings:

7315            *pMechanism*      pointer to the underlying cryptographic mechanism (e.g. AES, SHA-  
7316            256), see further 0, Appendix D

7317            *hKey*              handle to a key that will contribute to the entropy of the derived key  
7318            (CKM\_KIP\_DERIVE) or will be used in the MAC operation  
7319            (CKM\_KIP\_MAC)

7320            *pSeed*             pointer to an input seed

7321            *ulSeedLen*        length in bytes of the input seed

7322 **CK\_KIP\_PARAMS\_PTR** is a pointer to a **CK\_KIP\_PARAMS** structure.7323 

### 2.54.5 CT-KIP key derivation

7324 The CT-KIP key derivation mechanism, denoted **CKM\_KIP\_DERIVE**, is a key derivation mechanism that  
7325 is capable of generating secret keys of potentially any type, subject to token limitations.

7326 It takes a parameter of type **CK\_KIP\_PARAMS** which allows for the passing of the desired underlying  
 7327 cryptographic mechanism as well as some other data. In particular, when the *hKey* parameter is a handle  
 7328 to an existing key, that key will be used in the key derivation in addition to the *hBaseKey* of **C\_DeriveKey**.  
 7329 The *pSeed* parameter may be used to seed the key derivation operation.  
 7330 The mechanism derives a secret key with a particular set of attributes as specified in the attributes of the  
 7331 template for the key.  
 7332 The mechanism contributes the **CKA\_CLASS** and **CKA\_VALUE** attributes to the new key. Other  
 7333 attributes supported by the key type may be specified in the template for the key, or else will be assigned  
 7334 default initial values. Since the mechanism is generic, the **CKA\_KEY\_TYPE** attribute should be set in the  
 7335 template, if the key is to be used with a particular mechanism.

## 7336 **2.54.6 CT-KIP key wrap and key unwrap**

7337 The CT-KIP key wrap and unwrap mechanism, denoted **CKM\_KIP\_WRAP**, is a key wrap mechanism that  
 7338 is capable of wrapping and unwrapping generic secret keys.  
 7339 It takes a parameter of type **CK\_KIP\_PARAMS**, which allows for the passing of the desired underlying  
 7340 cryptographic mechanism as well as some other data. It does not make use of the *hKey* parameter of  
 7341 **CK\_KIP\_PARAMS**.

## 7342 **2.54.7 CT-KIP signature generation**

7343 The CT-KIP signature (MAC) mechanism, denoted **CKM\_KIP\_MAC**, is a mechanism used to produce a  
 7344 message authentication code of arbitrary length. The keys it uses are secret keys.  
 7345 It takes a parameter of type **CK\_KIP\_PARAMS**, which allows for the passing of the desired underlying  
 7346 cryptographic mechanism as well as some other data. The mechanism does not make use of the *pSeed*  
 7347 and the *uSeedLen* parameters of **CT\_KIP\_PARAMS**.  
 7348 This mechanism produces a MAC of the length specified by *pulSignatureLen* parameter in calls to  
 7349 **C\_Sign**.  
 7350 If a call to **C\_Sign** with this mechanism fails, then no output will be generated.

## 7351 **2.55 GOST 28147-89**

7352 GOST 28147-89 is a block cipher with 64-bit block size and 256-bit keys.  
 7353  
 7354 *Table 200, GOST 28147-89 Mechanisms vs. Functions*

| Mechanism              | Functions         |               |         |        |                    |               |        |
|------------------------|-------------------|---------------|---------|--------|--------------------|---------------|--------|
|                        | Encrypt & Decrypt | Sign & Verify | SR & VR | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_GOST28147_KEY_GEN  |                   |               |         |        | ✓                  |               |        |
| CKM_GOST28147_ECB      | ✓                 |               |         |        |                    | ✓             |        |
| CKM_GOST28147          | ✓                 |               |         |        |                    | ✓             |        |
| CKM_GOST28147_MAC      |                   | ✓             |         |        |                    |               |        |
| CKM_GOST28147_KEY_WRAP |                   |               |         |        |                    | ✓             |        |

7355

## 7356 2.55.1 Definitions

7357 This section defines the key type “CKK\_GOST28147” for type CK\_KEY\_TYPE as used in the  
7358 CKA\_KEY\_TYPE attribute of key objects and domain parameter objects.

7359 Mechanisms:

7360 CKM\_GOST28147\_KEY\_GEN  
7361 CKM\_GOST28147\_ECB  
7362 CKM\_GOST28147  
7363 CKM\_GOST28147\_MAC  
7364 CKM\_GOST28147\_KEY\_WRAP

## 7365 2.55.2 GOST 28147-89 secret key objects

7366 GOST 28147-89 secret key objects (object class **CKO\_SECRET\_KEY**, key type **CKK\_GOST28147**) hold  
7367 GOST 28147-89 keys. The following table defines the GOST 28147-89 secret key object attributes, in  
7368 addition to the common attributes defined for this object class:

7369 *Table 201, GOST 28147-89 Secret Key Object Attributes*

| Attribute                             | Data type  | Meaning                                                                                                                                                                                                                       |
|---------------------------------------|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup>          | Byte array | 32 bytes in little endian order                                                                                                                                                                                               |
| CKA_GOST28147_PARAMS <sup>1,3,5</sup> | Byte array | DER-encoding of the object identifier indicating the data object type of GOST 28147-89.<br><br>When key is used the domain parameter object of key type CKK_GOST28147 must be specified with the same attribute CKA_OBJECT_ID |

7370 Refer to [PKCS11-Base] Table 11 for footnotes

7371 The following is a sample template for creating a GOST 28147-89 secret key object:

```
7372 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
7373 CK_KEY_TYPE keyType = CKK_GOST28147;
7374 CK_UTF8CHAR label[] = "A GOST 28147-89 secret key object";
7375 CK_BYTE value[32] = {...};
7376 CK_BYTE params_oid[] = {0x06, 0x07, 0x2a, 0x85, 0x03, 0x02,
7377 0x02, 0x1f, 0x00};
7378 CK_BBOOL true = CK_TRUE;
7379 CK_ATTRIBUTE template[] = {
7380 {CKA_CLASS, &class, sizeof(class)},
7381 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
7382 {CKA_TOKEN, &true, sizeof(true)},
7383 {CKA_LABEL, label, sizeof(label)-1},
7384 {CKA_ENCRYPT, &true, sizeof(true)},
7385 {CKA_GOST28147_PARAMS, params_oid, sizeof(params_oid)},
7386 {CKA_VALUE, value, sizeof(value)}
7387 };
```

### 7388 2.55.3 GOST 28147-89 domain parameter objects

7389 GOST 28147-89 domain parameter objects (object class **CKO\_DOMAIN\_PARAMETERS**, key type  
7390 **CKK\_GOST28147**) hold GOST 28147-89 domain parameters.

7391 The following table defines the GOST 28147-89 domain parameter object attributes, in addition to the  
7392 common attributes defined for this object class:

7393 *Table 202, GOST 28147-89 Domain Parameter Object Attributes*

| Attribute                  | Data Type  | Meaning                                                                                                                      |
|----------------------------|------------|------------------------------------------------------------------------------------------------------------------------------|
| CKA_VALUE <sup>1</sup>     | Byte array | DER-encoding of the domain parameters as it was introduced in [4] section 8.1 (type <i>Gost28147-89-ParamSetParameters</i> ) |
| CKA_OBJECT_ID <sup>1</sup> | Byte array | DER-encoding of the object identifier indicating the domain parameters                                                       |

7394 Refer to [PKCS11-Base] Table 11 for footnotes

7395 For any particular token, there is no guarantee that a token supports domain parameters loading up  
7396 and/or fetching out. Furthermore, applications, that make direct use of domain parameters objects, should  
7397 take in account that **CKA\_VALUE** attribute may be inaccessible.

7398 The following is a sample template for creating a GOST 28147-89 domain parameter object:

```
7399 CK_OBJECT_CLASS class = CKO_DOMAIN_PARAMETERS;
7400 CK_KEY_TYPE keyType = CKK_GOST28147;
7401 CK_UTF8CHAR label[] = "A GOST 28147-89 cryptographic
7402 parameters object";
7403 CK_BYTE oid[] = {0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02,
7404 0x1f, 0x00};
7405 CK_BYTE value[] = {
7406 0x30, 0x62, 0x04, 0x40, 0x4c, 0xde, 0x38, 0x9c, 0x29, 0x89, 0xef, 0xb6,
7407 0xff, 0xeb, 0x56, 0xc5, 0x5e, 0xc2, 0x9b, 0x02, 0x98, 0x75, 0x61, 0x3b,
7408 0x11, 0x3f, 0x89, 0x60, 0x03, 0x97, 0x0c, 0x79, 0x8a, 0xa1, 0xd5, 0x5d,
7409 0xe2, 0x10, 0xad, 0x43, 0x37, 0x5d, 0xb3, 0x8e, 0xb4, 0x2c, 0x77, 0xe7,
7410 0xcd, 0x46, 0xca, 0xfa, 0xd6, 0x6a, 0x20, 0x1f, 0x70, 0xf4, 0x1e, 0xa4,
7411 0xab, 0x03, 0xf2, 0x21, 0x65, 0xb8, 0x44, 0xd8, 0x02, 0x01, 0x00, 0x02,
7412 0x01, 0x40, 0x30, 0x0b, 0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x0e,
7413 0x00, 0x05, 0x00
7414 };
7415 CK_BBOOL true = CK_TRUE;
7416 CK_ATTRIBUTE template[] = {
7417 {CKA_CLASS, &class, sizeof(class)},
7418 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
7419 {CKA_TOKEN, &true, sizeof(true)},
7420 {CKA_LABEL, label, sizeof(label)-1},
7421 {CKA_OBJECT_ID, oid, sizeof(oid)},
7422 {CKA_VALUE, value, sizeof(value)}
7423 };

```

### 7424 2.55.4 GOST 28147-89 key generation

7425 The GOST 28147-89 key generation mechanism, denoted **CKM\_GOST28147\_KEY\_GEN**, is a key  
7426 generation mechanism for GOST 28147-89.

7427 It does not have a parameter.

7428 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
7429 key. Other attributes supported by the GOST 28147-89 key type may be specified for objects of object  
7430 class **CKO\_SECRET\_KEY**.

7431 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** are not  
7432 used.

## 7433 2.55.5 GOST 28147-89-ECB

7434 GOST 28147-89-ECB, denoted **CKM\_GOST28147\_ECB**, is a mechanism for single and multiple-part  
7435 encryption and decryption; key wrapping; and key unwrapping, based on GOST 28147-89 and electronic  
7436 codebook mode.

7437 It does not have a parameter.

7438 This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to  
7439 wrap/unwrap every secret key that it supports.

7440 For wrapping (**C\_WrapKey**), the mechanism encrypts the value of the **CKA\_VALUE** attribute of the key  
7441 that is wrapped, padded on the trailing end with up to block size so that the resulting length is a multiple  
7442 of the block size.

7443 For unwrapping (**C\_UnwrapKey**), the mechanism decrypts the wrapped key, and truncates the result  
7444 according to the **CKA\_KEY\_TYPE** attribute of the template and, if it has one, and the key type supports  
7445 it, the **CKA\_VALUE\_LEN** attribute of the template. The mechanism contributes the result as the  
7446 **CKA\_VALUE** attribute of the new key.

7447 Constraints on key types and the length of data are summarized in the following table:

7448 *Table 203, GOST 28147-89-ECB: Key and Data Length*

| Function    | Key type      | Input length           | Output length                                     |
|-------------|---------------|------------------------|---------------------------------------------------|
| C_Encrypt   | CKK_GOST28147 | Multiple of block size | Same as input length                              |
| C_Decrypt   | CKK_GOST28147 | Multiple of block size | Same as input length                              |
| C_WrapKey   | CKK_GOST28147 | Any                    | Input length rounded up to multiple of block size |
| C_UnwrapKey | CKK_GOST28147 | Multiple of block size | Determined by type of key being unwrapped         |

7449

7450 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7451 are not used.

## 7452 2.55.6 GOST 28147-89 encryption mode except ECB

7453 GOST 28147-89 encryption mode except ECB, denoted **CKM\_GOST28147**, is a mechanism for single  
7454 and multiple-part encryption and decryption; key wrapping; and key unwrapping, based on  
7455 [GOST 28147-89] and CFB, counter mode, and additional CBC mode defined in [RFC 4357] section 2.  
7456 Encryption's parameters are specified in object identifier of attribute **CKA\_GOST28147\_PARAMS**.

7457 It has a parameter, which is an 8-byte initialization vector. This parameter may be omitted then a zero  
7458 initialization vector is used.

7459 This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to  
7460 wrap/unwrap every secret key that it supports.

7461 For wrapping (**C\_WrapKey**), the mechanism encrypts the value of the **CKA\_VALUE** attribute of the key  
7462 that is wrapped.

7463 For unwrapping (**C\_UnwrapKey**), the mechanism decrypts the wrapped key, and contributes the result as  
7464 the **CKA\_VALUE** attribute of the new key.

7465 Constraints on key types and the length of data are summarized in the following table:

7466 *Table 204, GOST 28147-89 encryption modes except ECB: Key and Data Length*

| Function    | Key type      | Input length | Output length                                                                                                                                                                                           |
|-------------|---------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| C_Encrypt   | CKK_GOST28147 | Any          | For counter mode and CFB is the same as input length. For CBC is the same as input length padded on the trailing end with up to block size so that the resulting length is a multiple of the block size |
| C_Decrypt   | CKK_GOST28147 | Any          |                                                                                                                                                                                                         |
| C_WrapKey   | CKK_GOST28147 | Any          |                                                                                                                                                                                                         |
| C_UnwrapKey | CKK_GOST28147 | Any          |                                                                                                                                                                                                         |

7467

7468 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7469 are not used.

## 7470 **2.55.7 GOST 28147-89-MAC**

7471 GOST 28147-89-MAC, denoted **CKM\_GOST28147\_MAC**, is a mechanism for data integrity and  
7472 authentication based on GOST 28147-89 and key meshing algorithms [RFC 4357] section 2.3.

7473 MACing parameters are specified in object identifier of attribute **CKA\_GOST28147\_PARAMS**.

7474 The output bytes from this mechanism are taken from the start of the final GOST 28147-89 cipher block  
7475 produced in the MACing process.

7476 It has a parameter, which is an 8-byte MAC initialization vector. This parameter may be omitted then a  
7477 zero initialization vector is used.

7478 Constraints on key types and the length of data are summarized in the following table:

7479 *Table 205, GOST28147-89-MAC: Key and Data Length*

| Function | Key type      | Data length | Signature length |
|----------|---------------|-------------|------------------|
| C_Sign   | CKK_GOST28147 | Any         | 4 bytes          |
| C_Verify | CKK_GOST28147 | Any         | 4 bytes          |

7480

7481 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7482 are not used.

7483

## 7484 **2.55.8 GOST 28147-89 keys wrapping/unwrapping with GOST 28147-89**

7485 GOST 28147-89 keys as a KEK (key encryption keys) for encryption GOST 28147-89 keys, denoted by  
7486 **CKM\_GOST28147\_KEY\_WRAP**, is a mechanism for key wrapping; and key unwrapping, based on  
7487 GOST 28147-89. Its purpose is to encrypt and decrypt keys have been generated by key generation  
7488 mechanism for GOST 28147-89.

7489 For wrapping (**C\_WrapKey**), the mechanism first computes MAC from the value of the **CKA\_VALUE**  
7490 attribute of the key that is wrapped and then encrypts in ECB mode the value of the **CKA\_VALUE**  
7491 attribute of the key that is wrapped. The result is 32 bytes of the key that is wrapped and 4 bytes of MAC.

7492 For unwrapping (**C\_UnwrapKey**), the mechanism first decrypts in ECB mode the 32 bytes of the key that  
7493 was wrapped and then computes MAC from the unwrapped key. Then compared together 4 bytes MAC

7494 has computed and 4 bytes MAC of the input. If these two MACs do not match the wrapped key is  
 7495 disallowed. The mechanism contributes the result as the **CKA\_VALUE** attribute of the unwrapped key.  
 7496 It has a parameter, which is an 8-byte MAC initialization vector. This parameter may be omitted then a  
 7497 zero initialization vector is used.  
 7498 Constraints on key types and the length of data are summarized in the following table:  
 7499 *Table 206, GOST 28147-89 keys as KEK: Key and Data Length*

| Function    | Key type      | Input length | Output length |
|-------------|---------------|--------------|---------------|
| C_WrapKey   | CKK_GOST28147 | 32 bytes     | 36 bytes      |
| C_UnwrapKey | CKK_GOST28147 | 32 bytes     | 36 bytes      |

7500  
 7501 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 7502 are not used.  
 7503

## 7504 **2.56 GOST R 34.11-94**

7505 GOST R 34.11-94 is a mechanism for message digesting, following the hash algorithm with 256-bit  
 7506 message digest defined in [GOST R 34.11-94].  
 7507

7508 *Table 207, GOST R 34.11-94 Mechanisms vs. Functions*

| Mechanism          | Functions         |               |         |        |                    |               |        |
|--------------------|-------------------|---------------|---------|--------|--------------------|---------------|--------|
|                    | Encrypt & Decrypt | Sign & Verify | SR & VR | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_GOSTR3411      |                   |               |         | ✓      |                    |               |        |
| CKM_GOSTR3411_HMAC |                   | ✓             |         |        |                    |               |        |

7509

### 7510 **2.56.1 Definitions**

7511 This section defines the key type "CKK\_GOSTR3411" for type CK\_KEY\_TYPE as used in the  
 7512 CKA\_KEY\_TYPE attribute of domain parameter objects.

7513 Mechanisms:

7514 CKM\_GOSTR3411  
 7515 CKM\_GOSTR3411\_HMAC

### 7516 **2.56.2 GOST R 34.11-94 domain parameter objects**

7517 GOST R 34.11-94 domain parameter objects (object class **CKO\_DOMAIN\_PARAMETERS**, key type  
 7518 **CKK\_GOSTR3411**) hold GOST R 34.11-94 domain parameters.

7519 The following table defines the GOST R 34.11-94 domain parameter object attributes, in addition to the  
 7520 common attributes defined for this object class:

7521 Table 208, GOST R 34.11-94 Domain Parameter Object Attributes

| Attribute                  | Data Type  | Meaning                                                                                                                      |
|----------------------------|------------|------------------------------------------------------------------------------------------------------------------------------|
| CKA_VALUE <sup>1</sup>     | Byte array | DER-encoding of the domain parameters as it was introduced in [4] section 8.2 (type <i>GostR3411-94-ParamSetParameters</i> ) |
| CKA_OBJECT_ID <sup>1</sup> | Byte array | DER-encoding of the object identifier indicating the domain parameters                                                       |

7522 Refer to [PKCS11-Base] Table 11 for footnotes

7523 For any particular token, there is no guarantee that a token supports domain parameters loading up  
 7524 and/or fetching out. Furthermore, applications, that make direct use of domain parameters objects, should  
 7525 take in account that **CKA\_VALUE** attribute may be inaccessible.

7526 The following is a sample template for creating a GOST R 34.11-94 domain parameter object:

```

7527 CK_OBJECT_CLASS class = CKO_DOMAIN_PARAMETERS;
7528 CK_KEY_TYPE keyType = CKK_GOSTR3411;
7529 CK_UTF8CHAR label[] = "A GOST R34.11-94 cryptographic
7530 parameters object";
7531 CK_BYTE oid[] = {0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02,
7532 0x1e, 0x00};
7533 CK_BYTE value[] = {
7534 0x30, 0x64, 0x04, 0x40, 0x4e, 0x57, 0x64, 0xd1, 0xab, 0x8d, 0xcb, 0xbf,
7535 0x94, 0x1a, 0x7a, 0x4d, 0x2c, 0xd1, 0x10, 0x10, 0xd6, 0xa0, 0x57, 0x35,
7536 0x8d, 0x38, 0xf2, 0xf7, 0x0f, 0x49, 0xd1, 0x5a, 0xea, 0x2f, 0x8d, 0x94,
7537 0x62, 0xee, 0x43, 0x09, 0xb3, 0xf4, 0xa6, 0xa2, 0x18, 0xc6, 0x98, 0xe3,
7538 0xc1, 0x7c, 0xe5, 0x7e, 0x70, 0x6b, 0x09, 0x66, 0xf7, 0x02, 0x3c, 0x8b,
7539 0x55, 0x95, 0xbf, 0x28, 0x39, 0xb3, 0x2e, 0xcc, 0x04, 0x20, 0x00, 0x00,
7540 0x00, 0x00,
7541 0x00, 0x00,
7542 0x00, 0x00, 0x00, 0x00, 0x00
7543 };
7544 CK_BBOOL true = CK_TRUE;
7545 CK_ATTRIBUTE template[] = {
7546 {CKA_CLASS, &class, sizeof(class)},
7547 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
7548 {CKA_TOKEN, &true, sizeof(true)},
7549 {CKA_LABEL, label, sizeof(label)-1},
7550 {CKA_OBJECT_ID, oid, sizeof(oid)},
7551 {CKA_VALUE, value, sizeof(value)}
7552 };

```

### 7553 2.56.3 GOST R 34.11-94 digest

7554 GOST R 34.11-94 digest, denoted **CKM\_GOSTR3411**, is a mechanism for message digesting based on  
 7555 GOST R 34.11-94 hash algorithm [GOST R 34.11-94].

7556 As a parameter this mechanism utilizes a DER-encoding of the object identifier. A mechanism parameter  
 7557 may be missed then parameters of the object identifier *id-GostR3411-94-CryptoProParamSet* [RFC 4357]  
 7558 (section 11.2) must be used.

7559 Constraints on the length of input and output data are summarized in the following table. For single-part  
 7560 digesting, the data and the digest may begin at the same location in memory.

7561 *Table 209, GOST R 34.11-94: Data Length*

| Function | Input length | Digest length |
|----------|--------------|---------------|
| C_Digest | Any          | 32 bytes      |

7562

7563 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7564 are not used.

## 7565 **2.56.4 GOST R 34.11-94 HMAC**

7566 GOST R 34.11-94 HMAC mechanism, denoted **CKM\_GOSTR3411\_HMAC**, is a mechanism for  
7567 signatures and verification. It uses the HMAC construction, based on the GOST R 34.11-94 hash  
7568 function [GOST R 34.11-94] and core HMAC algorithm [RFC 2104]. The keys it uses are of generic key  
7569 type **CKK\_GENERIC\_SECRET** or **CKK\_GOST28147**.

7570 To be conformed to GOST R 34.11-94 hash algorithm [GOST R 34.11-94] the block length of core HMAC  
7571 algorithm is 32 bytes long (see [RFC 2104] section 2, and [RFC 4357] section 3).

7572 As a parameter this mechanism utilizes a DER-encoding of the object identifier. A mechanism parameter  
7573 may be missed then parameters of the object identifier *id-GostR3411-94-CryptoProParamSet* [RFC 4357]  
7574 (section 11.2) must be used.

7575 Signatures (MACs) produced by this mechanism are of 32 bytes long.

7576 Constraints on the length of input and output data are summarized in the following table:

7577 *Table 210, GOST R 34.11-94 HMAC: Key And Data Length*

| Function | Key type                               | Data length | Signature length |
|----------|----------------------------------------|-------------|------------------|
| C_Sign   | CKK_GENERIC_SECRET or<br>CKK_GOST28147 | Any         | 32 byte          |
| C_Verify | CKK_GENERIC_SECRET or<br>CKK_GOST28147 | Any         | 32 bytes         |

7578 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7579 are not used.

## 7580 **2.57 GOST R 34.10-2001**

7581 GOST R 34.10-2001 is a mechanism for single- and multiple-part signatures and verification, following  
7582 the digital signature algorithm defined in [GOST R 34.10-2001].

7583

7584 *Table 211, GOST R34.10-2001 Mechanisms vs. Functions*

| Mechanism                    | Functions         |                |         |        |                    |               |        |
|------------------------------|-------------------|----------------|---------|--------|--------------------|---------------|--------|
|                              | Encrypt & Decrypt | Sign & Verify  | SR & VR | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_GOSTR3410_KEY_PAIR_GEN   |                   |                |         |        | ✓                  |               |        |
| CKM_GOSTR3410                |                   | ✓ <sup>1</sup> |         |        |                    |               |        |
| CKM_GOSTR3410_WITH_GOSTR3411 |                   | ✓              |         |        |                    |               |        |
| CKM_GOSTR3410_KEY_WRAP       |                   |                |         |        |                    | ✓             |        |
| CKM_GOSTR3410_DERIVE         |                   |                |         |        |                    |               | ✓      |

7585

1 Single-part operations only

7586

## 7587 2.57.1 Definitions

7588 This section defines the key type "CKK\_GOSTR3410" for type CK\_KEY\_TYPE as used in the  
7589 CKA\_KEY\_TYPE attribute of key objects and domain parameter objects.

7590 Mechanisms:

7591 CKM\_GOSTR3410\_KEY\_PAIR\_GEN  
7592 CKM\_GOSTR3410  
7593 CKM\_GOSTR3410\_WITH\_GOSTR3411  
7594 CKM\_GOSTR3410  
7595 CKM\_GOSTR3410\_KEY\_WRAP  
7596 CKM\_GOSTR3410\_DERIVE

## 7597 2.57.2 GOST R 34.10-2001 public key objects

7598 GOST R 34.10-2001 public key objects (object class **CKO\_PUBLIC\_KEY**, key type **CKK\_GOSTR3410**)  
7599 hold GOST R 34.10-2001 public keys.

7600 The following table defines the GOST R 34.10-2001 public key object attributes, in addition to the  
7601 common attributes defined for this object class:

7602 *Table 212, GOST R 34.10-2001 Public Key Object Attributes*

| Attribute                             | Data Type  | Meaning                                                                                                                                                                                                                                                       |
|---------------------------------------|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKA_VALUE <sup>1,4</sup>              | Byte array | 64 bytes for public key; 32 bytes for each coordinates X and Y of elliptic curve point P(X, Y) in little endian order                                                                                                                                         |
| CKA_GOSTR3410_PARAMS <sup>1,3</sup>   | Byte array | DER-encoding of the object identifier indicating the data object type of GOST R 34.10-2001.<br>When key is used the domain parameter object of key type CKK_GOSTR3410 must be specified with the same attribute CKA_OBJECT_ID                                 |
| CKA_GOSTR3411_PARAMS <sup>1,3,8</sup> | Byte array | DER-encoding of the object identifier indicating the data object type of GOST R 34.11-94.<br>When key is used the domain parameter object of key type CKK_GOSTR3411 must be specified with the same attribute CKA_OBJECT_ID                                   |
| CKA_GOST28147_PARAMS <sup>8</sup>     | Byte array | DER-encoding of the object identifier indicating the data object type of GOST 28147-89.<br>When key is used the domain parameter object of key type CKK_GOST28147 must be specified with the same attribute CKA_OBJECT_ID. The attribute value may be omitted |

7603 Refer to [PKCS11-Base] Table 11 for footnotes

7604 The following is a sample template for creating an GOST R 34.10-2001 public key object:

```

7605 CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
7606 CK_KEY_TYPE keyType = CKK_GOSTR3410;
7607 CK_UTF8CHAR label[] = "A GOST R34.10-2001 public key object";
7608 CK_BYTE gostR3410params_oid[] =
7609 {0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x23, 0x00};
7610 CK_BYTE gostR3411params_oid[] =
7611 {0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x00};
7612 CK_BYTE gost28147params_oid[] =
7613 {0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x00};
7614 CK_BYTE value[64] = {...};
7615 CK_BBOOL true = CK_TRUE;
7616 CK_ATTRIBUTE template[] = {
7617 {CKA_CLASS, &class, sizeof(class)},
7618 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
7619 {CKA_TOKEN, &true, sizeof(true)},
7620 {CKA_LABEL, label, sizeof(label)-1},
7621 {CKA_GOSTR3410_PARAMS, gostR3410params_oid,
7622 sizeof(gostR3410params_oid)},
7623 {CKA_GOSTR3411_PARAMS, gostR3411params_oid,
7624 sizeof(gostR3411params_oid)},
7625 {CKA_GOST28147_PARAMS, gost28147params_oid,
7626 sizeof(gost28147params_oid)},
7627 {CKA_VALUE, value, sizeof(value)}}
7628 };

```

### 7629 2.57.3 GOST R 34.10-2001 private key objects

7630 GOST R 34.10-2001 private key objects (object class **CKO\_PRIVATE\_KEY**, key type  
7631 **CKK\_GOSTR3410**) hold GOST R 34.10-2001 private keys.

7632 The following table defines the GOST R 34.10-2001 private key object attributes, in addition to the  
7633 common attributes defined for this object class:

7634 *Table 213, GOST R 34.10-2001 Private Key Object Attributes*

| Attribute                               | Data Type  | Meaning                                                                                                                                                                                                                           |
|-----------------------------------------|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup>            | Byte array | 32 bytes for private key in little endian order                                                                                                                                                                                   |
| CKA_GOSTR3410_PARAMS <sup>1,4,6</sup>   | Byte array | DER-encoding of the object identifier indicating the data object type of GOST R 34.10-2001.<br><br>When key is used the domain parameter object of key type CKK_GOSTR3410 must be specified with the same attribute CKA_OBJECT_ID |
| CKA_GOSTR3411_PARAMS <sup>1,4,6,8</sup> | Byte array | DER-encoding of the object identifier indicating the data object type of GOST R 34.11-94.<br><br>When key is used the domain parameter object of key type CKK_GOSTR3411 must be specified                                         |

| Attribute                             | Data Type  | Meaning                                                                                                                                                                                                                                                           |
|---------------------------------------|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                       |            | with the same attribute<br>CKA_OBJECT_ID                                                                                                                                                                                                                          |
| CKA_GOST28147_PARAMS <sup>4,6,8</sup> | Byte array | DER-encoding of the object identifier indicating the data object type of GOST 28147-89.<br><br>When key is used the domain parameter object of key type CKK_GOST28147 must be specified with the same attribute CKA_OBJECT_ID. The attribute value may be omitted |

7635 Refer to [PKCS11-Base] Table 11 for footnotes

7636 Note that when generating an GOST R 34.10-2001 private key, the GOST R 34.10-2001 domain  
7637 parameters are *not* specified in the key's template. This is because GOST R 34.10-2001 private keys are  
7638 only generated as part of an GOST R 34.10-2001 key pair, and the GOST R 34.10-2001 domain  
7639 parameters for the pair are specified in the template for the GOST R 34.10-2001 public key.

7640 The following is a sample template for creating an GOST R 34.10-2001 private key object:

```

7641 CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
7642 CK_KEY_TYPE keyType = CKK_GOSTR3410;
7643 CK_UTF8CHAR label[] = "A GOST R34.10-2001 private key
7644 object";
7645 CK_BYTE subject[] = {...};
7646 CK_BYTE id[] = {123};
7647 CK_BYTE gostR3410params_oid[] =
7648 {0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x23, 0x00};
7649 CK_BYTE gostR3411params_oid[] =
7650 {0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x00};
7651 CK_BYTE gost28147params_oid[] =
7652 {0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x00};
7653 CK_BYTE value[32] = {...};
7654 CK_BBOOL true = CK_TRUE;
7655 CK_ATTRIBUTE template[] = {
7656 {CKA_CLASS, &class, sizeof(class)},
7657 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
7658 {CKA_TOKEN, &true, sizeof(true)},
7659 {CKA_LABEL, label, sizeof(label)-1},
7660 {CKA_SUBJECT, subject, sizeof(subject)},
7661 {CKA_ID, id, sizeof(id)},
7662 {CKA_SENSITIVE, &true, sizeof(true)},
7663 {CKA_SIGN, &true, sizeof(true)},
7664 {CKA_GOSTR3410_PARAMS, gostR3410params_oid,
7665 sizeof(gostR3410params_oid)},
7666 {CKA_GOSTR3411_PARAMS, gostR3411params_oid,
7667 sizeof(gostR3411params_oid)},
7668 {CKA_GOST28147_PARAMS, gost28147params_oid,
7669 sizeof(gost28147params_oid)},
7670 {CKA_VALUE, value, sizeof(value)}}

```

7671 } ;  
7672

## 7673 2.57.4 GOST R 34.10-2001 domain parameter objects

7674 GOST R 34.10-2001 domain parameter objects (object class **CKO\_DOMAIN\_PARAMETERS**, key type  
7675 **CKK\_GOSTR3410**) hold GOST R 34.10-2001 domain parameters.

7676 The following table defines the GOST R 34.10-2001 domain parameter object attributes, in addition to the  
7677 common attributes defined for this object class:

7678 *Table 214, GOST R 34.10-2001 Domain Parameter Object Attributes*

| Attribute                  | Data Type  | Meaning                                                                                                                        |
|----------------------------|------------|--------------------------------------------------------------------------------------------------------------------------------|
| CKA_VALUE <sup>1</sup>     | Byte array | DER-encoding of the domain parameters as it was introduced in [4] section 8.4 (type <i>GostR3410-2001-ParamSetParameters</i> ) |
| CKA_OBJECT_ID <sup>1</sup> | Byte array | DER-encoding of the object identifier indicating the domain parameters                                                         |

7679 Refer to [PKCS11-Base] Table 11 for footnotes

7680 For any particular token, there is no guarantee that a token supports domain parameters loading up  
7681 and/or fetching out. Furthermore, applications, that make direct use of domain parameters objects, should  
7682 take in account that **CKA\_VALUE** attribute may be inaccessible.

7683 The following is a sample template for creating a GOST R 34.10-2001 domain parameter object:

```
7684 CK_OBJECT_CLASS class = CKO_DOMAIN_PARAMETERS;
7685 CK_KEY_TYPE keyType = CKK_GOSTR3410;
7686 CK_UTF8CHAR label[] = "A GOST R34.10-2001 cryptographic
7687 parameters object";
7688 CK_BYTE oid[] =
7689 {0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x23, 0x00};
7690 CK_BYTE value[] = {
7691 0x30, 0x81, 0x90, 0x02, 0x01, 0x07, 0x02, 0x20, 0x5f, 0xbff, 0xf4, 0x98,
7692 0xaa, 0x93, 0x8c, 0xe7, 0x39, 0xb8, 0xe0, 0x22, 0xfb, 0xaf, 0xef, 0x40,
7693 0x56, 0x3f, 0x6e, 0x6a, 0x34, 0x72, 0xfc, 0x2a, 0x51, 0x4c, 0x0c, 0xe9,
7694 0xda, 0xe2, 0x3b, 0x7e, 0x02, 0x21, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00,
7695 0x00, 0x00,
7696 0x00, 0x00,
7697 0x00, 0x04, 0x31, 0x02, 0x21, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00,
7698 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x50, 0xfe,
7699 0x8a, 0x18, 0x92, 0x97, 0x61, 0x54, 0xc5, 0x9c, 0xfc, 0x19, 0x3a, 0xcc,
7700 0xf5, 0xb3, 0x02, 0x01, 0x02, 0x20, 0x08, 0xe2, 0xa8, 0xa0, 0xe6,
7701 0x51, 0x47, 0xd4, 0xbd, 0x63, 0x16, 0x03, 0x0e, 0x16, 0xd1, 0x9c, 0x85,
7702 0xc9, 0x7f, 0x0a, 0x9c, 0xa2, 0x67, 0x12, 0x2b, 0x96, 0xab, 0xbc, 0xea,
7703 0x7e, 0x8f, 0xc8
7704 };
7705 CK_BBOOL true = CK_TRUE;
7706 CK_ATTRIBUTE template[] = {
7707 {CKA_CLASS, &class, sizeof(class)},
7708 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
7709 {CKA_TOKEN, &true, sizeof(true)},
7710 {CKA_LABEL, label, sizeof(label)-1},
```

```

7711 { CKA_OBJECT_ID, oid, sizeof(oid) },
7712 { CKA_VALUE, value, sizeof(value) }
7713 } ;
7714

```

## 7715 2.57.5 GOST R 34.10-2001 mechanism parameters

### 7716 ♦ CK\_GOSTR3410\_KEY\_WRAP\_PARAMS

7717 **CK\_GOSTR3410\_KEY\_WRAP\_PARAMS** is a structure that provides the parameters to the  
 7718 **CKM\_GOSTR3410\_KEY\_WRAP** mechanism. It is defined as follows:

```

7719 typedef struct CK_GOSTR3410_KEY_WRAP_PARAMS {
7720 CK_BYTE_PTR pWrapOID;
7721 CK ULONG ulWrapOIDLen;
7722 CK_BYTE_PTR pUKM;
7723 CK ULONG ulUKMLen;
7724 CK_OBJECT_HANDLE hKey;
7725 } CK_GOSTR3410_KEY_WRAP_PARAMS;
7726

```

7727 The fields of the structure have the following meanings:

|                     |                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>pWrapOID</i>     | pointer to a data with DER-encoding of the object identifier indicating the data object type of GOST 28147-89. If pointer takes NULL_PTR value in C_WrapKey operation then parameters are specified in object identifier of attribute CKA_GOSTR3411_PARAMS must be used. For C_UnwrapKey operation the pointer is not used and must take NULL_PTR value anytime |
| <i>ulWrapOIDLen</i> | length of data with DER-encoding of the object identifier indicating the data object type of GOST 28147-89                                                                                                                                                                                                                                                      |
| <i>pUKM</i>         | pointer to a data with UKM. If pointer takes NULL_PTR value in C_WrapKey operation then random value of UKM will be used. If pointer takes non-NUL_PTR value in C_UnwrapKey operation then the pointer value will be compared with UKM value of wrapped key. If these two values do not match the wrapped key will be rejected                                  |
| <i>ulUKMLen</i>     | length of UKM data. If <i>pUKM</i> -pointer is different from NULL_PTR then equal to 8                                                                                                                                                                                                                                                                          |
| <i>hKey</i>         | key handle. Key handle of a sender for C_WrapKey operation. Key handle of a receiver for C_UnwrapKey operation. When key handle takes CK_INVALID_HANDLE value then an ephemeral (one time) key pair of a sender will be used                                                                                                                                    |

7728 **CK\_GOSTR3410\_KEY\_WRAP\_PARAMS\_PTR** is a pointer to a  
 7729 **CK\_GOSTR3410\_KEY\_WRAP\_PARAMS**.

### 7730 ♦ CK\_GOSTR3410\_DERIVE\_PARAMS

7731 **CK\_GOSTR3410\_DERIVE\_PARAMS** is a structure that provides the parameters to the  
 7732 **CKM\_GOSTR3410\_DERIVE** mechanism. It is defined as follows:

```

7733 typedef struct CK_GOSTR3410_DERIVE_PARAMS {

```

```

7734 CK_EC_KDF_TYPE kdf;
7735 CK_BYTE_PTR pPublicData;
7736 CK ULONG ulPublicDataLen;
7737 CK_BYTE_PTR pUKM;
7738 CK ULONG ulUKMLen;
7739 } CK_GOSTR3410_DERIVE_PARAMS;
7740

```

7741 The fields of the structure have the following meanings:

|                                 |                                                                                                                                                                                                |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>kdf</i>                      | additional key diversification algorithm identifier.<br>Possible values are CKD_NULL and<br>CKD_CPDIVERSIFY_KDF. In case of CKD_NULL,<br>result of the key derivation function                 |
| <i>pPublicData</i> <sup>1</sup> | described in [RFC 4357], section 5.2 is used directly; In<br>case of CKD_CPDIVERSIFY_KDF, the resulting key<br>value is additionally processed with algorithm from [RFC<br>4357], section 6.5. |
| <i>ulPublicDataLen</i>          | pointer to data with public key of a receiver                                                                                                                                                  |
| <i>ulUKMLen</i>                 | length of data with public key of a receiver (must be 64)                                                                                                                                      |
| <i>pUKM</i>                     | pointer to a UKM data                                                                                                                                                                          |
| <i>ulUKMLen</i>                 | length of UKM data in bytes (must be 8)                                                                                                                                                        |

7742

7743 1 Public key of a receiver is an octet string of 64 bytes long. The public key octets correspond to the concatenation of X and Y coordinates of a point. Any one of  
7744 them is 32 bytes long and represented in little endian order.

7745 CK\_GOSTR3410\_DERIVE\_PARAMS\_PTR is a pointer to a CK\_GOSTR3410\_DERIVE\_PARAMS.

## 7746 2.57.6 GOST R 34.10-2001 key pair generation

7747 The GOST R 34.10-2001 key pair generation mechanism, denoted  
7748 **CKM\_GOSTR3410\_KEY\_PAIR\_GEN**, is a key pair generation mechanism for GOST R 34.10-2001.

7749 This mechanism does not have a parameter.

7750 The mechanism generates GOST R 34.10-2001 public/private key pairs with particular  
7751 GOST R 34.10-2001 domain parameters, as specified in the **CKA\_GOSTR3410\_PARAMS**,  
7752 **CKA\_GOSTR3411\_PARAMS**, and **CKA\_GOST28147\_PARAMS** attributes of the template for the public  
7753 key. Note that **CKA\_GOST28147\_PARAMS** attribute may not be present in the template.

7754 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
7755 public key and the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_VALUE**, and **CKA\_GOSTR3410\_PARAMS**,  
7756 **CKA\_GOSTR3411\_PARAMS**, **CKA\_GOST28147\_PARAMS** attributes to the new private key.

7757 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7758 are not used.

## 7759 2.57.7 GOST R 34.10-2001 without hashing

7760 The GOST R 34.10-2001 without hashing mechanism, denoted **CKM\_GOSTR3410**, is a mechanism for  
7761 single-part signatures and verification for GOST R 34.10-2001. (This mechanism corresponds only to the  
7762 part of GOST R 34.10-2001 that processes the 32-bytes hash value; it does not compute the hash value.)

7763 This mechanism does not have a parameter.

7764 For the purposes of these mechanisms, a GOST R 34.10-2001 signature is an octet string of 64 bytes  
7765 long. The signature octets correspond to the concatenation of the GOST R 34.10-2001 values  $s$  and  $r'$ ,  
7766 both represented as a 32 bytes octet string in big endian order with the most significant byte first [RFC  
7767 4490] section 3.2, and [RFC 4491] section 2.2.2.

7768 The input for the mechanism is an octet string of 32 bytes long with digest has computed by means of  
7769 GOST R 34.11-94 hash algorithm in the context of signed or should be signed message.

7770 *Table 215, GOST R 34.10-2001 without hashing: Key and Data Length*

| Function              | Key type      | Input length | Output length |
|-----------------------|---------------|--------------|---------------|
| C_Sign <sup>1</sup>   | CKK_GOSTR3410 | 32 bytes     | 64 bytes      |
| C_Verify <sup>1</sup> | CKK_GOSTR3410 | 32 bytes     | 64 bytes      |

7771 <sup>1</sup> Single-part operations only.

7772 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7773 are not used.

## 7774 **2.57.8 GOST R 34.10-2001 with GOST R 34.11-94**

7775 The GOST R 34.10-2001 with GOST R 34.11-94, denoted **CKM\_GOSTR3410\_WITH\_GOSTR3411**, is a  
7776 mechanism for signatures and verification for GOST R 34.10-2001. This mechanism computes the entire  
7777 GOST R 34.10-2001 specification, including the hashing with GOST R 34.11-94 hash algorithm.

7778 As a parameter this mechanism utilizes a DER-encoding of the object identifier indicating  
7779 GOST R 34.11-94 data object type. A mechanism parameter may be missed then parameters are  
7780 specified in object identifier of attribute **CKA\_GOSTR3411\_PARAMS** must be used.

7781 For the purposes of these mechanisms, a GOST R 34.10-2001 signature is an octet string of 64 bytes  
7782 long. The signature octets correspond to the concatenation of the GOST R 34.10-2001 values  $s$  and  $r'$ ,  
7783 both represented as a 32 bytes octet string in big endian order with the most significant byte first [RFC  
7784 4490] section 3.2, and [RFC 4491] section 2.2.2.

7785 The input for the mechanism is signed or should be signed message of any length. Single- and multiple-  
7786 part signature operations are available.

7787 *Table 216, GOST R 34.10-2001 with GOST R 34.11-94: Key and Data Length*

| Function | Key type      | Input length | Output length |
|----------|---------------|--------------|---------------|
| C_Sign   | CKK_GOSTR3410 | Any          | 64 bytes      |
| C_Verify | CKK_GOSTR3410 | Any          | 64 bytes      |

7788 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7789 are not used.

## 7790 **2.57.9 GOST 28147-89 keys wrapping/unwrapping with GOST R 34.10-2001**

7791 GOST R 34.10-2001 keys as a KEK (key encryption keys) for encryption GOST 28147 keys, denoted by  
7792 **CKM\_GOSTR3410\_KEY\_WRAP**, is a mechanism for key wrapping; and key unwrapping, based on  
7793 GOST R 34.10-2001. Its purpose is to encrypt and decrypt keys have been generated by key generation  
7794 mechanism for GOST 28147-89. An encryption algorithm from [RFC 4490] (section 5.2) must be used.  
7795 Encrypted key is a DER-encoded structure of ASN.1 *GostR3410-KeyTransport* type [RFC 4490] section  
7796 4.2.

7797 It has a parameter, a **CK\_GOSTR3410\_KEY\_WRAP\_PARAMS** structure defined in section 2.57.5.

7798 For unwrapping (**C\_UnwrapKey**), the mechanism decrypts the wrapped key, and contributes the result as  
7799 the **CKA\_VALUE** attribute of the new key.

7800 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7801 are not used.

## 7802 2.57.10 Common key derivation with assistance of GOST R 34.10-2001 keys

7803 Common key derivation, denoted **CKM\_GOSTR3410\_DERIVE**, is a mechanism for key derivation with  
7804 assistance of GOST R 34.10-2001 private and public keys. The key of the mechanism must be of object  
7805 class **CKO\_DOMAIN\_PARAMETERS** and key type **CKK\_GOSTR3410**. An algorithm for key derivation  
7806 from [RFC 4357] (section 5.2) must be used.

7807 The mechanism contributes the result as the **CKA\_VALUE** attribute of the new private key. All other  
7808 attributes must be specified in a template for creating private key object.

## 7809 2.58 ChaCha20

7810 ChaCha20 is a secret-key stream cipher described in [CHACHA].

7811 *Table 217, ChaCha20 Mechanisms vs. Functions*

| Mechanism            | Functions         |               |                      |        |                    |               |        |
|----------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                      | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_CHACHA20_KEY_GEN |                   |               |                      |        | ✓                  |               |        |
| CKM_CHACHA20         | ✓                 |               |                      |        |                    | ✓             |        |

7812

### 7813 2.58.1 Definitions

7814 This section defines the key type "CKK\_CHACHA20" for type CK\_KEY\_TYPE as used in the  
7815 CKA\_KEY\_TYPE attribute of key objects.

7816 Mechanisms:

7817 CKM\_CHACHA20\_KEY\_GEN

7818 CKM\_CHACHA20

### 7819 2.58.2 ChaCha20 secret key objects

7820 ChaCha20 secret key objects (object class CKO\_SECRET\_KEY, key type CKK\_CHACHA20) hold  
7821 ChaCha20 keys. The following table defines the ChaCha20 secret key object attributes, in addition to the  
7822 common attributes defined for this object class:

7823 *Table 218, ChaCha20 Secret Key Object*

| Attribute                    | Data type  | Meaning                                                                    |
|------------------------------|------------|----------------------------------------------------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup> | Byte array | Key length is fixed at 256 bits.<br>Bit length restricted to a byte array. |
| CKA_VALUE_LEN <sup>2,3</sup> | CK ULONG   | Length in bytes of key value                                               |

7824 The following is a sample template for creating a ChaCha20 secret key object:

```
7825 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
7826 CK_KEY_TYPE keyType = CKK_CHACHA20;
7827 CK_UTF8CHAR label[] = "A ChaCha20 secret key object";
7828 CK_BYTE value[32] = { ... };
7829 CK_BBOOL true = CK_TRUE;
7830 CK_ATTRIBUTE template[] = {
7831 {CKA_CLASS, &class, sizeof(class)},
```

```
7832 { CKA_KEY_TYPE, &keyType, sizeof(keyType) },
7833 { CKA_TOKEN, &true, sizeof(true) },
7834 { CKA_LABEL, label, sizeof(label)-1 },
7835 { CKA_ENCRYPT, &true, sizeof(true) },
7836 { CKA_VALUE, value, sizeof(value) }
7837 } ;
7838 CKA_CHECK_VALUE: The value of this attribute is derived from the key object by taking the first
7839 three bytes of the SHA-1 hash of the ChaCha20 secret key object's CKA_VALUE attribute.
```

### 7840 2.58.3 ChaCha20 mechanism parameters

#### 7841 ◆ CK\_CHACHA20\_PARAMS; CK\_CHACHA20\_PARAMS\_PTR

7842 **CK\_CHACHA20\_PARAMS** provides the parameters to the **CKM\_CHACHA20** mechanism. It is defined  
7843 as follows:

```
7844 typedef struct CK_CHACHA20_PARAMS {
7845 CK_BYTE_PTR pBlockCounter;
7846 CK ULONG blockCounterBits;
7847 CK_BYTE_PTR pNonce;
7848 CK ULONG ulNonceBits;
7849 } CK_CHACHA20_PARAMS;
```

7850 The fields of the structure have the following meanings:

|      |                           |                                                                                                |
|------|---------------------------|------------------------------------------------------------------------------------------------|
| 7851 | <i>pBlockCounter</i>      | <i>pointer to block counter</i>                                                                |
| 7852 | <i>ulblockCounterBits</i> | <i>length of block counter in bits (can be either 32 or 64)</i>                                |
| 7853 | <i>pNonce</i>             | <i>nonce (This should be never re-used with the same key.)</i>                                 |
| 7854 | <i>ulNonceBits</i>        | <i>length of nonce in bits (is 64 for original, 96 for IETF and 192 for xchacha20 variant)</i> |

7856 The block counter is used to address 512 bit blocks in the stream. In certain settings (e.g. disk encryption)  
7857 it is necessary to address these blocks in random order, thus this counter is exposed here.

7858 **CK\_CHACHA20\_PARAMS\_PTR** is a pointer to **CK\_CHACHA20\_PARAMS**.

### 7859 2.58.4 ChaCha20 key generation

7860 The ChaCha20 key generation mechanism, denoted **CKM\_CHACHA20\_KEY\_GEN**, is a key generation  
7861 mechanism for ChaCha20.

7862 It does not have a parameter.

7863 The mechanism generates ChaCha20 keys of 256 bits.

7864 The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new  
7865 key. Other attributes supported by the key type (specifically, the flags indicating which functions the key  
7866 supports) may be specified in the template for the key, or else are assigned default initial values.

7867 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7868 specify the supported range of key sizes in bytes. As a practical matter, the key size for ChaCha20 is  
7869 fixed at 256 bits.

7870

## 7871 2.58.5 ChaCha20 mechanism

7872 ChaCha20, denoted **CKM\_CHACHA20**, is a mechanism for single and multiple-part encryption and  
 7873 decryption based on the ChaCha20 stream cipher. It comes in 3 variants, which only differ in the size and  
 7874 handling of their nonces, affecting the safety of using random nonces and the maximum size that can be  
 7875 encrypted safely.

7876 Chacha20 has a parameter, **CK\_CHACHA20\_PARAMS**, which indicates the nonce and initial block  
 7877 counter value.

7878 Constraints on key types and the length of input and output data are summarized in the following table:

7879 *Table 219, ChaCha20: Key and Data Length*

| Function  | Key type | Input length                                    | Output length        | Comments      |
|-----------|----------|-------------------------------------------------|----------------------|---------------|
| C_Encrypt | ChaCha20 | Any / only up to 256 GB in case of IETF variant | Same as input length | No final part |
| C_Decrypt | ChaCha20 | Any / only up to 256 GB in case of IETF variant | Same as input length | No final part |

7880 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
 7881 specify the supported range of ChaCha20 key sizes, in bits.

7882 *Table 220, ChaCha20:Nonce and block counter lengths*

| Variant   | Nonce   | Block counter | Maximum message     | Nonce generation                                                                                      |
|-----------|---------|---------------|---------------------|-------------------------------------------------------------------------------------------------------|
| original  | 64 bit  | 64 bit        | Virtually unlimited | 1 <sup>st</sup> msg:<br>nonce <sub>0</sub> =random<br>n <sup>th</sup> msg:<br>nonce <sub>n-1</sub> ++ |
| IETF      | 96 bit  | 32 bit        | Max ~256 GB         | 1 <sup>st</sup> msg:<br>nonce <sub>0</sub> =random<br>n <sup>th</sup> msg:<br>nonce <sub>n-1</sub> ++ |
| XChaCha20 | 192 bit | 64 bit        | Virtually unlimited | Each nonce can be randomly generated.                                                                 |

7883 Nonces must not ever be reused with the same key. However due to the birthday paradox the first two  
 7884 variants cannot guarantee that randomly generated nonces are never repeating. Thus the recommended  
 7885 way to handle this is to generate the first nonce randomly, then increase this for follow-up messages.  
 7886 Only the last (XChaCha20) has large enough nonces so that it is virtually impossible to trigger with  
 7887 randomly generated nonces the birthday paradox.

## 7888 2.59 Salsa20

7889 Salsa20 is a secret-key stream cipher described in [SALSA].

7890 *Table 221, Salsa20 Mechanisms vs. Functions*

| Mechanism           | Functions         |               |                      |        |                    |               |        |
|---------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                     | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_SALSA20_KEY_GEN |                   |               |                      |        | ✓                  |               |        |
| CKM_SALSA20         | ✓                 |               |                      |        |                    | ✓             |        |

7891

### 7892 2.59.1 Definitions

7893 This section defines the key type "CKK\_SALSA20" and "CKK\_SALSA20" for type CK\_KEY\_TYPE as  
7894 used in the CKA\_KEY\_TYPE attribute of key objects.

7895 Mechanisms:

7896 CKM\_SALSA20\_KEY\_GEN

7897 CKM\_SALSA20

### 7898 2.59.2 Salsa20 secret key objects

7899 Salsa20 secret key objects (object class CKO\_SECRET\_KEY, key type CKK\_SALSA20) hold Salsa20  
7900 keys. The following table defines the Salsa20 secret key object attributes, in addition to the common  
7901 attributes defined for this object class:

7902 *Table 222, ChaCha20 Secret Key Object*

| Attribute                    | Data type  | Meaning                                                                    |
|------------------------------|------------|----------------------------------------------------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup> | Byte array | Key length is fixed at 256 bits.<br>Bit length restricted to a byte array. |
| CKA_VALUE_LEN <sup>2,3</sup> | CK ULONG   | Length in bytes of key value                                               |

7903 The following is a sample template for creating a Salsa20 secret key object:

```
7904 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
7905 CK_KEY_TYPE keyType = CKK_SALSA20;
7906 CK_UTF8CHAR label[] = "A Salsa20 secret key object";
7907 CK_BYTE value[32] = {...};
7908 CK_BBOOL true = CK_TRUE;
7909 CK_ATTRIBUTE template[] = {
7910 {CKA_CLASS, &class, sizeof(class)},
7911 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
7912 {CKA_TOKEN, &true, sizeof(true)},
7913 {CKA_LABEL, label, sizeof(label)-1},
7914 {CKA_ENCRYPT, &true, sizeof(true)},
7915 {CKA_VALUE, value, sizeof(value)}}
7916 };
```

7917 CKA\_CHECK\_VALUE: The value of this attribute is derived from the key object by taking the first  
7918 three bytes of the SHA-1 hash of the ChaCha20 secret key object's CKA\_VALUE attribute.

### 7919 2.59.3 Salsa20 mechanism parameters

#### 7920 ◆ CK\_SALSA20\_PARAMS; CK\_SALSA20\_PARAMS\_PTR

7921 CK\_SALSA20\_PARAMS provides the parameters to the CKM\_SALSA20 mechanism. It is defined as  
7922 follows:

```
7923 typedef struct CK_SALSA20_PARAMS {
7924 CK_BYTE_PTR pBlockCounter;
7925 CK_BYTE_PTR pNonce;
7926 CK ULONG ulNonceBits;
7927 } CK_SALSA20_PARAMS;
```

7928 The fields of the structure have the following meanings:

7929 pBlockCounter pointer to block counter (64 bits)

7931 pNonce nonce

7932 ulNonceBits size of the nonce in bits (64 for classic and 192 for XSalsa20)

7933 The block counter is used to address 512 bit blocks in the stream. In certain settings (e.g. disk encryption)  
7934 it is necessary to address these blocks in random order, thus this counter is exposed here.

7935 CK\_SALSA20\_PARAMS\_PTR is a pointer to CK\_SALSA20\_PARAMS.

### 7936 2.59.4 Salsa20 key generation

7937 The Salsa20 key generation mechanism, denoted CKM\_SALSA20\_KEY\_GEN, is a key generation  
7938 mechanism for Salsa20.

7939 It does not have a parameter.

7940 The mechanism generates Salsa20 keys of 256 bits.

7941 The mechanism contributes the CKA\_CLASS, CKA\_KEY\_TYPE, and CKA\_VALUE attributes to the new  
7942 key. Other attributes supported by the key type (specifically, the flags indicating which functions the key  
7943 supports) may be specified in the template for the key, or else are assigned default initial values.

7944 For this mechanism, the ulMinKeySize and ulMaxKeySize fields of the CK\_MECHANISM\_INFO structure  
7945 specify the supported range of key sizes in bytes. As a practical matter, the key size for Salsa20 is fixed  
7946 at 256 bits.

### 7947 2.59.5 Salsa20 mechanism

7948 Salsa20, denoted CKM\_SALSA20, is a mechanism for single and multiple-part encryption and decryption  
7949 based on the Salsa20 stream cipher. Salsa20 comes in two variants which only differ in the size and  
7950 handling of their nonces, affecting the safety of using random nonces.

7951 Salsa20 has a parameter, CK\_SALSA20\_PARAMS, which indicates the nonce and initial block counter  
7952 value.

7953 Constraints on key types and the length of input and output data are summarized in the following table:

7954 Table 223, Salsa20: Key and Data Length

| Function  | Key type | Input length | Output length        | Comments      |
|-----------|----------|--------------|----------------------|---------------|
| C_Encrypt | Salsa20  | Any          | Same as input length | No final part |
| C_Decrypt | Salsa20  | Any          | Same as input length | No final part |

7955 For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure  
7956 specify the supported range of ChaCha20 key sizes, in bits.

7957 *Table 224, Salsa20: Nonce sizes*

| Variant  | Nonce   | Maximum message     | Nonce generation                                                                                      |
|----------|---------|---------------------|-------------------------------------------------------------------------------------------------------|
| original | 64 bit  | Virtually unlimited | 1 <sup>st</sup> msg:<br>nonce <sub>0</sub> =random<br>n <sup>th</sup> msg:<br>nonce <sub>n-1</sub> ++ |
| XSalsa20 | 192 bit | Virtually unlimited | Each nonce can be randomly generated.                                                                 |

7958 Nonces must not ever be reused with the same key. However due to the birthday paradox the original  
7959 variant cannot guarantee that randomly generated nonces are never repeating. Thus the recommended  
7960 way to handle this is to generate the first nonce randomly, then increase this for follow-up messages.  
7961 Only the XSalsa20 has large enough nonces so that it is virtually impossible to trigger with randomly  
7962 generated nonces the birthday paradox.

## 7963 **2.60 Poly1305**

7964 Poly1305 is a message authentication code designed by D.J Bernstein [**POLY1305**]. Poly1305 takes a  
7965 256 bit key and a message and produces a 128 bit tag that is used to verify the message.

7966 *Table 225, Poly1305 Mechanisms vs. Functions*

| Mechanism            | Functions         |               |                      |        |                    |               |        |
|----------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                      | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_POLY1305_KEY_GEN |                   |               |                      |        | ✓                  |               |        |
| CKM_POLY1305         |                   | ✓             |                      |        |                    |               |        |

### 7967 **2.60.1 Definitions**

7968 This section defines the key type “CKK\_POLY1305” for type CK\_KEY\_TYPE as used in the  
7969 CKA\_KEY\_TYPE attribute of key objects.

7970 Mechanisms:

7971       CKM\_POLY1305\_KEY\_GEN

7972       CKM\_POLY1305

7973 **2.60.2 Poly1305 secret key objects**

7974 Poly1305 secret key objects (object class CKO\_SECRET\_KEY, key type CKK\_POLY1305) hold  
7975 Poly1305 keys. The following table defines the Poly1305 secret key object attributes, in addition to the  
7976 common attributes defined for this object class:

7977 *Table 226, Poly1305 Secret Key Object*

| Attribute                    | Data type  | Meaning                                                                    |
|------------------------------|------------|----------------------------------------------------------------------------|
| CKA_VALUE <sup>1,4,6,7</sup> | Byte array | Key length is fixed at 256 bits.<br>Bit length restricted to a byte array. |
| CKA_VALUE_LEN <sup>2,3</sup> | CK ULONG   | Length in bytes of key value                                               |

7978 The following is a sample template for creating a Poly1305 secret key object:

```
7979 CK_OBJECT_CLASS class = CKO_SECRET_KEY;
7980 CK_KEY_TYPE keyType = CKK_POLY1305;
7981 CK_UTF8CHAR label[] = "A Poly1305 secret key object";
7982 CK_BYTE value[32] = {...};
7983 CK_BBOOL true = CK_TRUE;
7984 CK_ATTRIBUTE template[] = {
7985 {CKA_CLASS, &class, sizeof(class)},
7986 {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
7987 {CKA_TOKEN, &true, sizeof(true)},
7988 {CKA_LABEL, label, sizeof(label)-1},
7989 {CKA_SIGN, &true, sizeof(true)},
7990 {CKA_VALUE, value, sizeof(value)}}
7991 };
7992
```

7993 **2.60.3 Poly1305 mechanism**

7994 Poly1305, denoted **CKM\_POLY1305**, is a mechanism for producing an output tag based on a 256 bit key  
7995 and arbitrary length input.

7996 It has no parameters.

7997 Signatures (MACs) produced by this mechanism will be fixed at 128 bits in size.

7998 *Table 227, Poly1305: Key and Data Length*

| Function | Key type | Data length | Signature Length |
|----------|----------|-------------|------------------|
| C_Sign   | Poly1305 | Any         | 128 bits         |
| C_Verify | Poly1305 | Any         | 128 bits         |

7999 **2.61 Chacha20/Poly1305 and Salsa20/Poly1305 Authenticated  
8000 Encryption / Decryption**

8001 The stream ciphers Salsa20 and ChaCha20 are normally used in conjunction with the Poly1305  
8002 authenticator, in such a construction they also provide Authenticated Encryption with Associated Data  
8003 (AEAD). This section defines the combined mechanisms and their usage in an AEAD setting.

8004 *Table 228, Poly1305 Mechanisms vs. Functions*

| Mechanism             | Functions         |               |                      |        |                    |               |        |
|-----------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                       | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_CHACHA20_POLY1305 | ✓                 |               |                      |        |                    |               |        |
| CKM_SALSA20_POLY1305  | ✓                 |               |                      |        |                    |               |        |

## 8005    2.61.1 Definitions

8006    Mechanisms:

8007       CKM\_CHACHA20\_POLY1305

8008       CKM\_SALSA20\_POLY1305

## 8009    2.61.2 Usage

8010    Generic ChaCha20, Salsa20, Poly1305 modes are described in [CHACHA], [SALSA] and [POLY1305].  
 8011    To set up for ChaCha20/Poly1305 or Salsa20/Poly1305 use the following process. ChaCha20/Poly1305  
 8012    and Salsa20/Poly1305 both use CK\_SALSA20\_CHACHA20\_POLY1305\_PARAMS for Encrypt, Decrypt  
 8013    and CK\_SALSA20\_CHACHA20\_POLY1305\_MSG\_PARAMS for MessageEncrypt, and MessageDecrypt.

8014    Encrypt:

- Set the Nonce length *uNonceLen* in the parameter block. (this affects which variant of Chacha20 will be used: 64 bits → original, 96 bits → IETF, 192 bits → XChaCha20)
- Set the Nonce data *pNonce* in the parameter block.
- Set the AAD data *pAAD* and size *uAADLen* in the parameter block. *pAAD* may be NULL if *uAADLen* is 0.
- Call C\_EncryptInit() for **CKM\_CHACHA20\_POLY1305** or **CKM\_SALSA20\_POLY1305** mechanism with parameters and key *K*.
- Call C\_Encrypt(), or C\_EncryptUpdate()<sup>\*10</sup> C\_EncryptFinal(), for the plaintext obtaining ciphertext and authentication tag output.

8024    Decrypt:

- Set the Nonce length *uNonceLen* in the parameter block. (this affects which variant of Chacha20 will be used: 64 bits → original, 96 bits → IETF, 192 bits → XChaCha20)
- Set the Nonce data *pNonce* in the parameter block.
- Set the AAD data *pAAD* and size *uAADLen* in the parameter block. *pAAD* may be NULL if *uAADLen* is 0.
- Call C\_DecryptInit() for **CKM\_CHACHA20\_POLY1305** or **CKM\_SALSA20\_POLY1305** mechanism with parameters and key *K*.
- Call C\_Decrypt(), or C\_DecryptUpdate()<sup>\*1</sup> C\_DecryptFinal(), for the ciphertext, including the appended tag, obtaining plaintext output. Note: since **CKM\_CHACHA20\_POLY1305** and **CKM\_SALSA20\_POLY1305** are AEAD ciphers, no data should be returned until C\_Decrypt() or C\_DecryptFinal().

---

10 “\*” indicates 0 or more calls may be made as required

8036 MessageEncrypt::

- 8037     • Set the Nonce length *uNonceLen* in the parameter block. (this affects which variant of Chacha20  
8038       will be used: 64 bits → original, 96 bits → IETF, 192 bits → XChaCha20)
- 8039     • Set the Nonce data *pNonce* in the parameter block.
- 8040     • Set *pTag* to hold the tag data returned from *C\_EncryptMessage()* or the final  
8041       *C\_EncryptMessageNext()*.
- 8042     • Call *C\_MessageEncryptInit()* for **CKM\_CHACHA20\_POLY1305** or **CKM\_SALSA20\_POLY1305**  
8043       mechanism with key *K*.
- 8044     • Call *C\_EncryptMessage()*, or *C\_EncryptMessageBegin* followed by *C\_EncryptMessageNext()*\*<sup>11</sup>.  
8045       The mechanism parameter is passed to all three of these functions.
- 8046     • Call *C\_MessageEncryptFinal()* to close the message decryption.

8047 MessageDecrypt:

- 8048     • Set the Nonce length *uNonceLen* in the parameter block. (this affects which variant of Chacha20  
8049       will be used: 64 bits → original, 96 bits → IETF, 192 bits → XChaCha20)
- 8050     • Set the Nonce data *pNonce* in the parameter block.
- 8051     • Set the tag data *pTag* in the parameter block before *C\_DecryptMessage* or the final  
8052       *C\_DecryptMessageNext()*
- 8053     • Call *C\_MessageDecryptInit()* for **CKM\_CHACHA20\_POLY1305** or **CKM\_SALSA20\_POLY1305**  
8054       mechanism with key *K*.
- 8055     • Call *C\_DecryptMessage()*, or *C\_DecryptMessageBegin* followed by *C\_DecryptMessageNext()*\*<sup>12</sup>.  
8056       The mechanism parameter is passed to all three of these functions.
- 8057     • Call *C\_MessageDecryptFinal()* to close the message decryption

8058

8059     *uNonceLen* is the length of the nonce in bits.

8060     In Encrypt and Decrypt the tag is appended to the cipher text. In MessageEncrypt the tag is returned in  
8061       the *pTag* field of *CK\_SALSA20\_CHACHA20\_POLY1305\_MSG\_PARAMS*. In MessageDecrypt the tag is  
8062       provided by the *pTag* field of *CK\_SALSA20\_CHACHA20\_POLY1305\_MSG\_PARAMS*. The application  
8063       must provide 16 bytes of space for the tag.

8064     The key type for *K* must be compatible with **CKM\_CHACHA20** or **CKM\_SALSA20** respectively and the  
8065       *C\_EncryptInit/C\_DecryptInit* calls shall behave, with respect to *K*, as if they were called directly with  
8066       **CKM\_CHACHA20** or **CKM\_SALSA20**, *K* and NULL parameters.

8067     Unlike the atomic Salsa20/ChaCha20 mechanism the AEAD mechanism based on them does not expose  
8068       the block counter, as the AEAD construction is based on a message metaphor in which random access is  
8069       not needed.

## 8070     2.61.3 ChaCha20/Poly1305 and Salsa20/Poly1305 Mechanism parameters

8071     ◆ **CK\_SALSA20\_CHACHA20\_POLY1305\_PARAMS;**  
8072       **CK\_SALSA20\_CHACHA20\_POLY1305\_PARAMS\_PTR**

8073     **CK\_SALSA20\_CHACHA20\_POLY1305\_PARAMS** is a structure that provides the parameters to the  
8074       **CKM\_CHACHA20\_POLY1305** and **CKM\_SALSA20\_POLY1305** mechanisms. It is defined as follows:

---

11 “\*” indicates 0 or more calls may be made as required

12 “\*” indicates 0 or more calls may be made as required

```

8075 typedef struct CK_SALSA20_CHACHA20_POLY1305_PARAMS {
8076 CK_BYTE_PTR pNonce;
8077 CK ULONG ulNonceLen;
8078 CK_BYTE_PTR pAAD;
8079 CK ULONG ulAADLen;
8080 } CK_SALSA20_CHACHA20_POLY1305_PARAMS;
8081
8082 The fields of the structure have the following meanings:
8083 pNonce nonce (This should be never re-used with the same key.)
8084 ulNonceLen length of nonce in bits (is 64 for original, 96 for IETF (only for
8085 chacha20) and 192 for xchacha20/xsalsa20 variant)
8086 pAAD pointer to additional authentication data. This data is authenticated
8087 but not encrypted.
8088 ulAADLen length of pAAD in bytes.
8089
8090 CK_SALSA20_CHACHA20_POLY1305_PARAMS_PTR is a pointer to a
8091 CK_SALSA20_CHACHA20_POLY1305_PARAMS.
8092
8093 ♦ CK_SALSA20_CHACHA20_POLY1305_MSG_PARAMS;
8094 CK_SALSA20_CHACHA20_POLY1305_MSG_PARAMS_PTR
8095
8096 CK_CHACHA20POLY1305_PARAMS is a structure that provides the parameters to the CKM_
8097 CHACHA20_POLY1305 mechanism. It is defined as follows:
8098
8099 typedef struct CK_SALSA20_CHACHA20_POLY1305_MSG_PARAMS {
8100 CK_BYTE_PTR pNonce;
8101 CK ULONG ulNonceLen;
8102 CK_BYTE_PTR pTag;
8103 } CK_SALSA20_CHACHA20_POLY1305_MSG_PARAMS;
8104
8105 The fields of the structure have the following meanings:
8106 pNonce pointer to nonce
8107 ulNonceLen length of nonce in bits. The length of the influences which variant of
8108 the ChaCha20 will be used (64 original, 96 IETF(only for
8109 ChaCha20), 192 XChaCha20/XSalsa20)
8110
8111 pTag location of the authentication tag which is returned on
8112 MessageEncrypt, and provided on MessageDecrypt.
8113
8114 CK_SALSA20_CHACHA20_POLY1305_MSG_PARAMS_PTR is a pointer to a
8115 CK_SALSA20_CHACHA20_POLY1305_MSG_PARAMS.
8116
8117 2.62 HKDF Mechanisms
8118
8119 Details for HKDF key derivation mechanisms can be found in [RFC 5869].
8120
8121 Table 229, HKDF Mechanisms vs. Functions

```

| Mechanism        | Functions         |               |                      |        |                    |               |        |
|------------------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|                  | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_HKDF_DERIVE  |                   |               |                      |        |                    |               | ✓      |
| CKM_HKDF_DATA    |                   |               |                      |        |                    |               | ✓      |
| CKM_HKDF_KEY_GEN |                   |               |                      |        | ✓                  |               |        |

## 8112 2.62.1 Definitions

8113 Mechanisms:

8114 CKM\_HKDF\_DERIVE  
 8115 CKM\_HKDF\_DATA  
 8116 CKM\_HKDF\_KEY\_GEN

8117

8118 Key Types:

8119 CKK\_HKDF

## 8120 2.62.2 HKDF mechanism parameters

### 8121 ♦ CK\_HKDF\_PARAMS; CK\_HKDF\_PARAMS\_PTR

8122 CK\_HKDF\_PARAMS is a structure that provides the parameters to the CKM\_HKDF\_DERIVE and CKM\_HKDF\_DATA mechanisms. It is defined as follows:

```
8124 typedef struct CK_HKDF_PARAMS {
8125 CK_BBOOL bExtract;
8126 CK_BBOOL bExpand;
8127 CK_MECHANISM_TYPE prfHashMechanism;
8128 CK ULONG ulSaltType;
8129 CK_BYTE_PTR pSalt;
8130 CK ULONG ulSaltLen;
8131 CK_OBJECT_HANDLE hSaltKey;
8132 CK_BYTE_PTR pInfo;
8133 CK ULONG ulInfoLen;
8134 } CK_HKDF_PARAMS;
```

8135

8136 The fields of the structure have the following meanings:

8137 *bExtract* execute the extract portion of HKDF.

8138 *bExpand* execute the expand portion of HKDF.

8139 *prfHashMechanism* base hash used for the HMAC in the underlying HKDF operation.

8140 *ulSaltType* specifies how the salt for the extract portion of the KDF is supplied.

8141 *CKF\_HKDF\_SALT\_NULL* no salt is supplied.

8142 *CKF\_HKDF\_SALT\_DATA* salt is supplied as a data in *pSalt* with length *ulSaltLen*.

8144                             *CKF\_HKDF\_SALT\_KEY salt is supplied as a key in hSaltKey.*  
8145                     *pSalt*     *pointer to the salt.*  
8146                     *ulSaltLen*     *length of the salt pointed to in pSalt.*  
8147                     *hSaltKey*     *object handle to the salt key.*  
8148                     *pInfo*     *info string for the expand stage.*  
8149                     *ullInfoLen*     *length of the info string for the expand stage.*

8150

8151 **CK\_HKDF\_PARAMS\_PTR** is a pointer to a **CK\_HKDF\_PARAMS**.

### 8152 2.62.3 HKDF derive

8153 HKDF derivation implements the HKDF as specified in [RFC 5869]. The two booleans *bExtract* and  
8154 *bExpand* control whether the extract section of the HKDF or the expand section of the HKDF is in use.  
8155 It has a parameter, a **CK\_HKDF\_PARAMS** structure, which allows for the passing of the salt and or the  
8156 expansion info. The structure contains the bools *bExtract* and *bExpand* which control whether the extract  
8157 or expand portions of the HKDF is to be used. This structure is defined in Section 2.62.2.

8158 The input key must be of type **CKK\_HKDF** or **CKK\_GENERIC\_SECRET** and the length must be the size  
8159 of the underlying hash function specified in *prfHashMechanism*. The exception is a data object which has  
8160 the same size as the underlying hash function, and which may be supplied as an input key. In this case  
8161 *bExtract* should be true and non-null salt should be supplied.

8162 Either *bExtract* or *bExpand* must be set to true. If they are both set to true, input key is first extracted then  
8163 expanded. The salt is used in the extraction stage. If *bExtract* is set to true and no salt is given, a 'zero'  
8164 salt (salt whose length is the same as the underlying hash and values all set to zero) is used as specified  
8165 by the RFC. If *bExpand* is set to true, **CKA\_VALUE\_LEN** should be set to the desired key length. If it is  
8166 false **CKA\_VALUE\_LEN** may be set to the length of the hash, but that is not necessary as the mechanism  
8167 will supply this value. The salt should be ignored if *bExtract* is false. The *pInfo* should be ignored if  
8168 *bExpand* is set to false.

8169 The mechanism also contributes the **CKA\_CLASS**, and **CKA\_VALUE** attributes to the new key. Other  
8170 attributes may be specified in the template, or else are assigned default values.

8171 The template sent along with this mechanism during a **C\_DeriveKey** call may indicate that the object  
8172 class is **CKO\_SECRET\_KEY**. However, since these facts are all implicit in the mechanism, there is no  
8173 need to specify any of them.

8174 This mechanism has the following rules about key sensitivity and extractability:

- The **CKA\_SENSITIVE** and **CKA\_EXTRACTABLE** attributes in the template for the new key can both  
8176 be specified to be either CK\_TRUE or CK\_FALSE. If omitted, these attributes each take on some  
8177 default value.
- If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_FALSE, then the derived key  
8179 will as well. If the base key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to CK\_TRUE, then the  
8180 derived key has its **CKA\_ALWAYS\_SENSITIVE** attribute set to the same value as its  
8181 **CKA\_SENSITIVE** attribute.
- Similarly, if the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to CK\_FALSE, then the  
8183 derived key will, too. If the base key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to  
8184 CK\_TRUE, then the derived key has its **CKA\_NEVER\_EXTRACTABLE** attribute set to the *opposite*  
8185 value from its **CKA\_EXTRACTABLE** attribute.

8186 **2.62.4 HKDF Data**

8187 HKDF Data derive mechanism, denoted **CKM\_HKDF\_DATA**, is identical to HKDF Derive except the  
8188 output is a **CKO\_DATA** object whose value is the result to the derive operation. Some tokens may restrict  
8189 what data may be successfully derived based on the pInfo portion of the CK\_HKDF\_PARAMS. All tokens  
8190 must minimally support bExtract set to true and pInfo values which contain the value "tls1.3 iv" as opaque  
8191 label as per [TLS13] struct HkdfLabel. Future additional required combinations may be specified in the  
8192 profile document and applications could then query the appropriate profile before depending on the  
8193 mechanism.

8194 **2.62.5 HKDF Key gen**

8195 HKDF key gen, denoted CKM\_HKDF\_KEY\_GEN generates a new random HKDF key.  
8196 CKA\_VALUE\_LEN must be set in the template.

8197 **2.63 NULL Mechanism**

8198 **CKM\_NULL** is a mechanism used to implement the trivial pass-through function.

8199

8200 *Table 230, CKM\_NULL Mechanisms vs. Functions*

| Mechanism | Functions         |               |                      |        |                    |               |        |
|-----------|-------------------|---------------|----------------------|--------|--------------------|---------------|--------|
|           | Encrypt & Decrypt | Sign & Verify | SR & VR <sup>1</sup> | Digest | Gen. Key/ Key Pair | Wrap & Unwrap | Derive |
| CKM_NULL  | ✓                 | ✓             | ✓                    | ✓      |                    | ✓             | ✓      |

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

8201

8202 **2.63.1 Definitions**

8203 Mechanisms:

8204       CKM\_NULL

8205 **2.63.2 CKM\_NULL mechanism parameters**

8206 CKM\_NULL does not have a parameter.

8207

8208 When used for encrypting / decrypting data, the input data is copied unchanged to the output data.

8209 When used for signing, the input data is copied to the signature. When used for signature verification, it  
8210 compares the input data and the signature, and returns CKR\_OK (indicating that both are identical) or  
8211 CKR\_SIGNATURE\_INVALID.

8212 When used for digesting data, the input data is copied to the message digest.

8213 When used for wrapping a private or secret key object, the wrapped key will be identical to the key to be  
8214 wrapped. When used for unwrapping, a new object with the same value as the wrapped key will be  
8215 created.

8216 When used for deriving a key, the derived key has the same value as the base key.

---

8217

## 3 PKCS #11 Implementation Conformance

8218 An implementation is a conforming implementation if it meets the conditions specified in one or more  
8219 server profiles specified in [PKCS11-Prof].

8220 If a PKCS #11 implementation claims support for a particular profile, then the implementation SHALL  
8221 conform to all normative statements within the clauses specified for that profile and for any subclauses to  
8222 each of those clauses.

---

## 8223 Appendix A. Acknowledgments

8224 The following individuals have participated in the creation of this specification and are gratefully  
8225 acknowledged:

8226 **Participants:**

8227 Benton Stark - Cisco Systems  
8228 Anthony Berglas - Cryptsoft Pty Ltd.  
8229 Justin Corlett - Cryptsoft Pty Ltd.  
8230 Tony Cox - Cryptsoft Pty Ltd.  
8231 Tim Hudson - Cryptsoft Pty Ltd.  
8232 Bruce Rich - Cryptsoft Pty Ltd.  
8233 Greg Scott - Cryptsoft Pty Ltd.  
8234 Jason Thatcher - Cryptsoft Pty Ltd.  
8235 Magda Zdunkiewicz - Cryptsoft Pty Ltd.  
8236 Andrew Byrne - Dell  
8237 David Horton - Dell  
8238 Kevin Mooney - Fornetix  
8239 Gerald Stueve - Fornetix  
8240 Charles White - Fornetix  
8241 Matt Bauer - Galois, Inc.  
8242 Wan-Teh Chang - Google Inc.  
8243 Patrick Steuer - IBM  
8244 Michele Drgon - Individual  
8245 Gershon Janssen - Individual  
8246 Oscar So - Individual  
8247 Michelle Brochmann - Information Security Corporation  
8248 Michael Mrkowitz - Information Security Corporation  
8249 Jonathan Schulze-Hewett - Information Security Corporation  
8250 Philip Lafrance - ISARA Corporation  
8251 Thomas Hardjono - M.I.T.  
8252 Hamish Cameron - nCipher  
8253 Paul King - nCipher  
8254 Sander Temme - nCipher  
8255 Chet Ensign - OASIS  
8256 Jane Harnad - OASIS  
8257 Web Master - OASIS  
8258 Dee Schur - OASIS  
8259 Xuelei Fan - Oracle  
8260 Jan Friedel - Oracle  
8261 Susan Gleeson - Oracle  
8262 Dina Kurktchi-Nimeh - Oracle

8263 John Leser - Oracle  
8264 Darren Moffat - Oracle  
8265 Mark Joseph - P6R, Inc  
8266 Jim Susoy - P6R, Inc  
8267 Roland Bramm - PrimeKey Solutions AB  
8268 Warren Armstrong - QuintessenceLabs Pty Ltd.  
8269 Kenli Chong - QuintessenceLabs Pty Ltd.  
8270 John Leiseboer - QuintessenceLabs Pty Ltd.  
8271 Florian Poppa - QuintessenceLabs Pty Ltd.  
8272 Martin Shannon - QuintessenceLabs Pty Ltd.  
8273 Jakub Jelen - Red Hat  
8274 Chris Malafis - Red Hat  
8275 Robert Relyea - Red Hat  
8276 Christian Bollich - Utimaco IS GmbH  
8277 Dieter Bong - Utimaco IS GmbH  
8278 Chris Meyer - Utimaco IS GmbH  
8279 Daniel Minder - Utimaco IS GmbH  
8280 Roland Reichenberg - Utimaco IS GmbH  
8281 Manish Upasani - Utimaco IS GmbH  
8282 Steven Wierenga - Utimaco IS GmbH

---

## 8283    **Appendix B. Manifest Constants**

8284    The definitions for manifest constants specified in this document can be found in the following normative  
8285    computer language definition files:

- 8286        • [include/pkcs11-v3.00/pkcs11.h](#)
- 8287        • [include/pkcs11-v3.00/pkcs11t.h](#)
- 8288        • [include/pkcs11-v3.00/pkcs11f.h](#)

---

8290

## Appendix C. Revision History

| Revision               | Date       | Editor                     | Changes Made                                      |
|------------------------|------------|----------------------------|---------------------------------------------------|
| csprd 02<br>wd01       | Oct 2 2019 | Dieter Bong                | Created csprd02 based on csprd01                  |
| csprd 02<br>wd02 .. 04 |            | Dieter Bong, Daniel Minder | Intermediate versions                             |
| csprd 02<br>wd05       | Dec 3 2019 | Dieter Bong, Daniel Minder | Changes as per "PKCS11 mechanisms review-v9.docx" |

8291