Intel® QuickAssist Technology Cryptographic API Reference

Automatically generated from sources, April 10, 2018.

Based on API version 2.1

(See Release Notes to map API version to software package version.)

Reference Number: 330685-005

Reference Number: 330685-005

Revision History

Date	Revision	Description
April 2017	005	Added session update API.
July 2016	004	Added Intel® Key Protection Technology (KPT) API.
October 2015	003	Changed version of the crypto API to v2.0. Added ZUC-EEA3 and ZUC-EIA3 support to the crypto API. Added SHA3-256 support to the crypto API.
September 2015	002	Incrementing CY API version number to v1.9. Adding CPA_STATUS_UNSUPPORTED as a return status for each function and callback.
June 2014	001	First “public” version of the document. Based on “Intel Confidential” document number 410923-1.8 with the revision history of that document retained for reference purposes.
April 2014	1.8	· Fixing specification of __FreeBSD__ · Whitespace clean-up · IXA00384099: Adding default 'None' entries to CpaCySymOp and CpaCySymHashAlgorithm · IXA00384099: Addition of CPA_CY_SYM_HASH_AES_CBC_MAC · IXA00384492: Addition of cpaCySymSessionCtxGetDynamicSize() and cpaCySymDpSessionCtxGetDynamicSize() · IXA00385073: Added performance guidance notes for source buffer lengths on the crypto API.
February 2013	1.7	Addition of AES-XTS mode
November 2012	1.6-RC2	Resolves the following work requests: · TECG00000192: Complete AES-GMAC support
October 2012	1.5	Resolves the following work requests: · TECG00000186: Add instance notification support for RESTARTING & RESTARTED events and CPA_STATUS_RESTARTING return codes.
October 2012	1.6-RC1	Resolves the following work requests: · TECG00000187: Add support for AES-F8 · TECG00000189: Add a unique instance identifier to CpaInstanceInfo2
June 2012	1.4	Resolves comments against previous revision. Resolves the following work requests: · TECG00000178: Removing CPA_CY_KEY_GEN_SSL_TLS_SEED_LEN_IN_BYTES from

		cpa_cy_key.h · TECG00000180: Adding detail on GMAC to API comments. · TECG00000181: Update RSA comments to call out no padding. · TECG00000182: DSA FIPS PUB 186-2 with Change Notice 1 updates to supported DSA key lengths. · TECG00000183: Clarifying that the message buffers may not be cleared when using the DP API if digest verification fails for CCM/GCM.
May 2012	1.3	Resolves the following work requests: · TECG00000175: Add support partial packets for chained operations and nested hash operations. · TECG00000162: Removed references to digestVerify and updated description of pDigestResult. · TECG00000167: cpaCyDhKeyGenPhase1 does not generate private value (x) on CCK
Apr 2012	1.3-RC15	Resolves the following work requests: · TECG00000169: Removing CPA_CY_SYM_DP_TMP_WORKAROUND from cpa_cy_sym_dp.h · TECG00000170: (IXA00372445) Updated API comments to say that it is safe to assume that cpaCySymDpSessionCtxGetSize() will always return the same size for a given implementation. Same for cpaCySymSessionCtxGetSize().
Mar 2012	1.3-RC14	Resolves the following work requests: · TECG00000166: Added ability to query bus address information for a CpaInstance.
Nov 2011	1.3-RC13	Resolved comments against RC12.
Oct 2011	1.3-RC12	Resolves the following work requests: · TECG00000135: Updated comments on key generation API with references to RFC5246 (TLS v1.2) · TECG00000147: Added hashAlgorithm parameter to TLS v1.2 PRF function · TECG00000153: Clarified cases when digest result should point to src vs. dst buffer · TECG00000154: Documented that verification failure for GCM/CCM will not result in the buffer being zeroised. Also added flag on DP API to indicate whether digestIsEncrypted. · TECG00000155: Removed parameter (number of requests submitted) from "perform op now" function · TECG00000156: Documented that some "unused" fields are in fact reserved for internal usage.

Jul 2011	1.3-RC11	Updated DP API per feedback from engineering during implementation
Jun 2011	1-3-RC10	Resolves comments against previous revisions, including the “traditional” and data plane APIs. Also includes updates for the following work requests: · TECG00000119: clarified max length for aadLenInBytes · TECG00000120: added support for 512-bit RSA operations · TECG00000121: added support for TLS 1.2 PRF/key generation function · TECG00000082: added support for batch submission of requests (via data plane API) · TECG00000030: clarified how large numbers are represented on the API
Apr 2011	1.3-RC8	Adds the data plane API for symmetric crypto, specifically file cpa_cy_sym_dp.h. Also adds new types to represent flag buffers and buffer lists with physical addressing.
Apr 2011	1.3-RC9	Resolves the following issues/work requests: · TECG00000098: drbg: Clarified description of reseed counter. · TECG00000108: keygen: Updated description of MGF function to refer to PKCS#1 MGF1 function. Also added @ref to some Doxygen comments to prettify the documentation. · TECG00000101: nrbg: Clarified that length of requested entropy must be >0 · TECG00000097: prime: updated the list of bit-sizes of prime number candidates supported · TECG00000117: Updated description of various fields for GCM and CCM, specifically to allow these algorithms to be implemented entirely underneath the API and therefore enabling the implementations to be FIPS certified under CAVP *Note: Data Plane API has been removed from this revision, updates based on previous review and this review will be incorporated in the next revision of the API.*
Sep 2010	1.3-RC7	Resolves the following issues/work requests: · TECG00000086, “DH API constraints on exponent need to be clarified” – removed offending sentences · TECG00000090, “Consider making some CY stats use 64-bit counters” – deprecated 32-bit counters on “legacy” APIs, added 64-bit counter support everywhere · Added a symmetric-specific “capability” to specify whether partial packets are supported on a given API instance/implementation
Mar 2010	1.3-RC5	Documents version 1.3 Release Candidate #5 of the API, incorporating feedback from the formal review. Key changes:

		· Removed point compression API (pending requirement) · Updated DSA API with support for FIPS 186-3 · Made DRBG reseed function asynchronous and clarified context constraints on this API · Numerous other minor clean-ups, clarifications, etc. · Added CPA_STATUS_UNSUPPORTED return code to the base API, to be returned when an implementation does not support a given capability.
Mar 2010	1.3-RC6	Corrected signature of DRBG session init function to include separate callback function pointers for Generate and Reseed functionality. Also tidied up this revision history table.
Dec 2009	1.3-RC4	Documents version 1.3 Release Candidate #4 of the API · TECG00000068: Merged minor changes from EP80579 · TECG00000069: ECDSA verify – removed input parameter · TECG00000047: Updated DSA to support FIPS 186-3 · TECG00000048: MGF hash function now configurable · TECG00000050: Added point decompaction to Elliptic Curve API · TECG00000062: Corrected comment re "authenticated cipher" on session setup data structure · TECG00000066: Clarified that partial packet is not supported for Kasumi & SNOW3G · TECG00000067: Clarified documentation of digestResultLenInBytes · TECG00000076: Clarified that for GCM/CCM decrypt, digestVerify is ignored · TECG00000081: Updated DRBG and NRBG APIs based on feedback from Hifn · TECG00000085: Resolve tech pubs feedback on QA CY API v1.3-RC3
Sep 2009	1.3-RC3	Documents version 1.3 Release Candidate #3 of the API, incorporating feedback from the formal review. Key changes: · On the RBG API, renamed a DRBG “instance” to a “session” (to avoid confusion with other instances and for consistency with symmetric sessions). Also fixed signature of the reseed function, and clarified some comments. · For elliptic curve crypto, clarified some comments. · Made crypto capabilities more granular. · Fixed some @context tags.

		· Fixed some typos in doxygen @ref tags. · Marked all deprecated functions/types so that they generate warnings when used. · Fixed definitions of TRUE and FALSE. · Added extern "C" linkage to all header files for C++ compilers. · Replaced all tabs with spaces for consistent indentation.
July 2009	1.3-RC2	Documents version 1.3 Release Candidate #2 of the API · Base API updated to reflect the decisions around Instances · Incorporates feedback from the informal review of v1.3-RC1 · TECG37: Clarified parameter usage for RSA KeyGen · TECG11: Clarified documentation around the enum CpaCyKeyTlsOp
June 2009	1.3-RC1	Documents version 1.3 Release Candidate #1 of the API · Incorporates the new cipher and authentication algorithms for wireless (Kasumi F8/F9, SNOW3G UEA2/UIA2, AES-CMAC). This was inherited from engineering with minor changes (addition of AES-CMAC, renaming of KGCORE to F8, etc.). · TECG17, TECG27: Incorporates the new elliptic curve algorithms. This was inherited from engineering with some minor changes (removed review comments/resolutions, renamed field types, etc.) · TECG29: Incorporates the changes to DRBG/NRBG to allow for certification. The old random APIs have been deprecated. · TECG25: Adds "capabilities". Two levels are added: one to indicate which sub-API groups are supported; and for symmetric, one to say which "optional" ciphers are supported. · Merged some changes due to IXA WRs: all comment changes (e.g. addition of RETRY return status from QueryStats functions on some APIs, and other minor clarification text.
July 2008	1.1	First released version of this document. Documents version 1.1 of the API.

Table of Contents

1 Deprecated List **1**

2 CPA API **3**
2.1 Detailed Description 3
2.2 Modules 3

3 Base Data Types [CPA API] **4**
3.1 Detailed Description 4
3.2 Data Structures 4
3.3 Defines 4
3.4 Typedefs 5
3.5 Enumerations 5
3.6 Data Structure Documentation 5
3.6.1 _CpaFlatBuffer Struct Reference 6
3.6.2 _CpaBufferList Struct Reference 6
3.6.3 _CpaPhysFlatBuffer Struct Reference 7
3.6.4 _CpaPhysBufferList Struct Reference 8
3.6.5 _CpaInstanceInfo Struct Reference 9
3.6.6 _CpaPhysicalInstanceId Struct Reference 10
3.6.7 _CpaInstanceInfo2 Struct Reference 11
3.7 Define Documentation 13
3.8 Typedef Documentation 15
3.9 Enumeration Type Documentation 19

4 CPA Type Definition [CPA API] **21**
4.1 Detailed Description 21
4.2 Defines 21
4.3 Typedefs 21
4.4 Enumerations 21
4.5 Define Documentation 21
4.6 Typedef Documentation 23
4.7 Enumeration Type Documentation 24

5 Cryptographic API [CPA API] **25**
5.1 Detailed Description 26
5.2 Modules 26

6 Cryptographic Common API [Cryptographic API] **27**
6.1 Detailed Description 27
6.2 Typedefs 27
6.3 Enumerations 27
6.4 Functions 27
6.5 Typedef Documentation 28
6.6 Enumeration Type Documentation 30
6.7 Function Documentation 31

7 Cryptographic Instance Management API [Cryptographic API] **39**
7.1 Detailed Description 39
7.2 Data Structures 39
7.3 Typedefs 39
7.4 Functions 39
7.5 Data Structure Documentation 39
7.5.1 _CpaCyCapabilitiesInfo Struct Reference 39
7.6 Typedef Documentation 41
7.7 Function Documentation 41

Table of Contents

8 Symmetric Cipher and Hash Cryptographic API [Cryptographic API]....46
8.1 Detailed Description....46
8.2 Modules....46
8.3 Data Structures....46
8.4 Defines....46
8.5 Typedefs....46
8.6 Enumerations....47
8.7 Functions....48
8.8 Data Structure Documentation....48
8.8.1 _CpaCySymCipherSetupData Struct Reference....48
8.8.2 _CpaCySymHashNestedModeSetupData Struct Reference....49
8.8.3 _CpaCySymHashAuthModeSetupData Struct Reference....50
8.8.4 _CpaCySymHashSetupData Struct Reference....51
8.8.5 _CpaCySymSessionSetupData Struct Reference....53
8.8.6 _CpaCySymSessionUpdateData Struct Reference....55
8.8.7 _CpaCySymOpData Struct Reference....55
8.8.8 _CpaCySymStats Struct Reference....59
8.8.9 _CpaCySymStats64 Struct Reference....60
8.8.10 _CpaCySymCapabilitiesInfo Struct Reference....61
8.9 Define Documentation....62
8.10 Typedef Documentation....63
8.11 Enumeration Type Documentation....67
8.12 Function Documentation....72

9 Symmetric cryptographic Data Plane API [Symmetric Cipher and Hash Cryptographic API]....85
9.1 Detailed Description....85
9.2 Data Structures....86
9.3 Typedefs....86
9.4 Functions....86
9.5 Data Structure Documentation....86
9.5.1 _CpaCySymDpOpData Struct Reference....86
9.6 Typedef Documentation....90
9.7 Function Documentation....92

10 Cryptographic Key and Mask Generation API [Cryptographic API]....102
10.1 Detailed Description....102
10.2 Data Structures....102
10.3 Defines....102
10.4 Typedefs....102
10.5 Enumerations....102
10.6 Functions....103
10.7 Data Structure Documentation....103
10.7.1 _CpaCyKeyGenSslOpData Struct Reference....103
10.7.2 _CpaCyKeyGenTlsOpData Struct Reference....105
10.7.3 _CpaCyKeyGenMgfOpData Struct Reference....107
10.7.4 _CpaCyKeyGenMgfOpDataExt Struct Reference....108
10.7.5 _CpaCyKeyGenStats Struct Reference....109
10.7.6 _CpaCyKeyGenStats64 Struct Reference....110
10.8 Define Documentation....111
10.9 Typedef Documentation....112
10.10 Enumeration Type Documentation....115
10.11 Function Documentation....116

Table of Contents

11 RSA API [Cryptographic API]....126

11.1 Detailed Description....126
11.2 Data Structures....126
11.3 Typedefs....126
11.4 Enumerations....127
11.5 Functions....127
11.6 Data Structure Documentation....127
11.6.1 _CpaCyRsaPublicKey Struct Reference....127
11.6.2 _CpaCyRsaPrivateKeyRep1 Struct Reference....128
11.6.3 _CpaCyRsaPrivateKeyRep2 Struct Reference....129
11.6.4 _CpaCyRsaPrivateKey Struct Reference....130
11.6.5 _CpaCyRsaKeyGenOpData Struct Reference....132
11.6.6 _CpaCyRsaEncryptOpData Struct Reference....134
11.6.7 _CpaCyRsaDecryptOpData Struct Reference....135
11.6.8 _CpaCyRsaStats Struct Reference....137
11.6.9 _CpaCyRsaStats64 Struct Reference....138
11.7 Typedef Documentation....140
11.8 Enumeration Type Documentation....143
11.9 Function Documentation....144

12 Diffie-Hellman (DH) API [Cryptographic API]....151

12.1 Detailed Description....151
12.2 Data Structures....151
12.3 Typedefs....151
12.4 Functions....151
12.5 Data Structure Documentation....152
12.5.1 _CpaCyDhPhase1KeyGenOpData Struct Reference....152
12.5.2 _CpaCyDhPhase2SecretKeyGenOpData Struct Reference....153
12.5.3 _CpaCyDhStats Struct Reference....154
12.5.4 _CpaCyDhStats64 Struct Reference....155
12.6 Typedef Documentation....156
12.7 Function Documentation....157

13 Digital Signature Algorithm (DSA) API [Cryptographic API]....163

13.1 Detailed Description....163
13.2 Data Structures....163
13.3 Typedefs....164
13.4 Functions....164
13.5 Data Structure Documentation....164
13.5.1 _CpaCyDsaPParamGenOpData Struct Reference....165
13.5.2 _CpaCyDsaGParamGenOpData Struct Reference....166
13.5.3 _CpaCyDsaYParamGenOpData Struct Reference....167
13.5.4 _CpaCyDsaRSignOpData Struct Reference....168
13.5.5 _CpaCyDsaSSignOpData Struct Reference....169
13.5.6 _CpaCyDsaRSSignOpData Struct Reference....171
13.5.7 _CpaCyDsaVerifyOpData Struct Reference....173
13.5.8 _CpaCyDsaStats Struct Reference....174
13.5.9 _CpaCyDsaStats64 Struct Reference....177
13.6 Typedef Documentation....180
13.7 Function Documentation....186

14 Elliptic Curve (EC) API [Cryptographic API]....199

14.1 Detailed Description....199
14.2 Data Structures....199
14.3 Typedefs....199

Table of Contents

14 Elliptic Curve (EC) API [Cryptographic API]
14.4 Enumerations....199
14.5 Functions....200
14.6 Data Structure Documentation....200
14.6.1 _CpaCyEcPointMultiplyOpData Struct Reference....200
14.6.2 _CpaCyEcPointVerifyOpData Struct Reference....202
14.6.3 _CpaCyEcStats64 Struct Reference....203
14.7 Typedef Documentation....204
14.8 Enumeration Type Documentation....207
14.9 Function Documentation....208

15 Elliptic Curve Diffie-Hellman (ECDH) API [Cryptographic API]....214
15.1 Detailed Description....214
15.2 Data Structures....214
15.3 Typedefs....214
15.4 Functions....214
15.5 Data Structure Documentation....214
15.5.1 _CpaCyEcdhPointMultiplyOpData Struct Reference....215
15.5.2 _CpaCyEcdhStats64 Struct Reference....216
15.6 Typedef Documentation....217
15.7 Function Documentation....219

16 Elliptic Curve Digital Signature Algorithm (ECDSA) API [Cryptographic API]....222
16.1 Detailed Description....222
16.2 Data Structures....222
16.3 Typedefs....222
16.4 Functions....222
16.5 Data Structure Documentation....223
16.5.1 _CpaCyEcdsaSignROpData Struct Reference....223
16.5.2 _CpaCyEcdsaSignSOpData Struct Reference....225
16.5.3 _CpaCyEcdsaSignRSOpData Struct Reference....226
16.5.4 _CpaCyEcdsaVerifyOpData Struct Reference....229
16.5.5 _CpaCyEcdsaStats64 Struct Reference....231
16.6 Typedef Documentation....233
16.7 Function Documentation....237

17 Cryptographic Large Number API [Cryptographic API]....244
17.1 Detailed Description....244
17.2 Data Structures....244
17.3 Typedefs....244
17.4 Functions....245
17.5 Data Structure Documentation....245
17.5.1 _CpaCyLnModExpOpData Struct Reference....245
17.5.2 _CpaCyLnModInvOpData Struct Reference....246
17.5.3 _CpaCyLnStats Struct Reference....247
17.5.4 _CpaCyLnStats64 Struct Reference....248
17.6 Typedef Documentation....249
17.7 Function Documentation....250

18 Prime Number Test API [Cryptographic API]....256
18.1 Detailed Description....256
18.2 Data Structures....256
18.3 Typedefs....256
18.4 Functions....256
18.5 Data Structure Documentation....256

Table of Contents

18 Prime Number Test API [Cryptographic API]
18.5.1 _CpaCyPrimeTestOpData Struct Reference....257
18.5.2 _CpaCyPrimeStats Struct Reference....258
18.5.3 _CpaCyPrimeStats64 Struct Reference....259
18.6 Typedef Documentation....260
18.7 Function Documentation....262

19 Deterministic Random Bit Generation API [Cryptographic API]....264
19.1 Detailed Description....264
19.2 Data Structures....264
19.3 Typedefs....264
19.4 Enumerations....264
19.5 Functions....265
19.6 Data Structure Documentation....265
19.6.1 _CpaCyDrbgSessionSetupData Struct Reference....265
19.6.2 _CpaCyDrbgGenOpData Struct Reference....266
19.6.3 _CpaCyDrbgReseedOpData Struct Reference....267
19.6.4 _CpaCyDrbgStats64 Struct Reference....268
19.7 Typedef Documentation....269
19.8 Enumeration Type Documentation....271
19.9 Function Documentation....271

20 Non-Deterministic Random Bit Generation API [Cryptographic API]....279
20.1 Detailed Description....279
20.2 Data Structures....279
20.3 Typedefs....279
20.4 Functions....279
20.5 Data Structure Documentation....279
20.5.1 _CpaCyNrbgOpData Struct Reference....279
20.6 Typedef Documentation....280
20.7 Function Documentation....280

21 Random Bit/Number Generation API [Cryptographic API]....282
21.1 Detailed Description....282
21.2 Data Structures....282
21.3 Defines....282
21.4 Typedefs....282
21.5 Functions....282
21.6 Data Structure Documentation....282
21.6.1 _CpaCyRandStats Struct Reference....283
21.6.2 _CpaCyRandGenOpData Struct Reference....284
21.6.3 _CpaCyRandSeedOpData Struct Reference....285
21.7 Define Documentation....286
21.8 Typedef Documentation....286
21.9 Function Documentation....287

22 Intel(R) Key Protection Technology (KPT) Cryptographic API [Cryptographic API]....292
22.1 Detailed Description....292
22.2 Data Structures....292
22.3 Defines....292
22.4 Typedefs....292
22.5 Enumerations....293
22.6 Functions....293
22.7 Data Structure Documentation....294
22.7.1 CpaCyKptWrappingFormat_t Struct Reference....294

Table of Contents

22 Intel(R) Key Protection Technology (KPT) Cryptographic API [Cryptographic API]
22.7.2 CpaCyKptRsaWpkSizeRep2_t Struct Reference....295
22.7.3 CpaCyKptWpkSize_t Union Reference....295
22.7.4 CpaCyKptUnwrapContext_t Struct Reference....296
22.7.5 _CpaCyKptEcdsaSignRSOpData Struct Reference....298
22.8 Define Documentation....300
22.9 Typedef Documentation....301
22.10 Enumeration Type Documentation....303
22.11 Function Documentation....305

1 Deprecated List

Class **_CpaCyDhStats**
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyDhStats64**.

Class **_CpaCyDsaStats**
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyDsaStats64**.

Class **_CpaCyKeyGenStats**
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyKeyGenStats64**.

Class **_CpaCyLnStats**
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyLnStats64**.

Class **_CpaCyPrimeStats**
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyPrimeStats64**.

Class **_CpaCyRandGenOpData**
As of v1.3 of the API, replaced by **CpaCyDrbgGenOpData**.

Class **_CpaCyRandSeedOpData**
As of v1.3 of the API, replaced by **CpaCyDrbgReseedOpData**.

Class **_CpaCyRandStats**
As of v1.3 of the API, replaced by **CpaCyDrbgStats64**.

Class **_CpaCyRsaStats**
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyRsaStats64**.

Class **_CpaCySymStats**
As of v1.3 of the cryptographic API, this structure has been deprecated, replaced by **CpaCySymStats64**.

Class **_CpaInstanceInfo**
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by CpaInstanceInfo2.

Global **CPA_DEPRECATED**
As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaAccelerationServiceType**.

Global **CPA_DEPRECATED**
As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaOperationalState**.

1 Deprecated List

Global **cpaCyInstanceGetInfo**
: As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyInstanceGetInfo2**.

Global **cpaCySymQueryStats**
: As of v1.3 of the cryptographic API, this function has been deprecated, replaced by **cpaCySymQueryStats64()**.

Global **cpaCyKeyGenQueryStats**
: As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyKeyGenQueryStats64()**.

Global **cpaCyRsaQueryStats**
: As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyRsaQueryStats64()**.

Global **cpaCyDhQueryStats**
: As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyDhQueryStats64()**.

Global **cpaCyDsaQueryStats**
: As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyDsaQueryStats64()**.

Global **cpaCyLnStatsQuery**
: As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyLnStatsQuery64()**.

Group **cpaCyRand**
: As of v1.3 of the API, this entire API group has been deprecated, replaced by API groups **Deterministic Random Bit Generation API** and **Non-Deterministic Random Bit Generation API** .

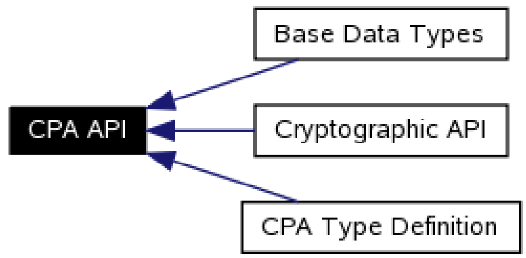
Global **cpaCyRandGen**
: As of v1.3 of the API, replaced by **cpaCyDrbgGen()**.

Global **cpaCyRandSeed**
: As of v1.3 of the API, replaced by **cpaCyDrbgReseed()**.

Global **cpaCyRandQueryStats**
: As of v1.3 of the API, replaced by **cpaCyDrbgQueryStats64()**.

2 CPA API

Collaboration diagram for CPA API:



2.1 Detailed Description

File: cpa.h

This is the top level API definition for Intel(R) QuickAssist Technology. It contains structures, data types and definitions that are common across the interface.

2.2 Modules

- **Base Data Types**
- **CPA Type Definition**
- **Cryptographic API**

3 Base Data Types

[CPA API]

Collaboration diagram for Base Data Types:

3.1 Detailed Description

File: cpa.h

The base data types for the Intel CPA API.

3.2 Data Structures

- struct **_CpaFlatBuffer**
- struct **_CpaBufferList**
- struct **_CpaPhysFlatBuffer**
- struct **_CpaPhysBufferList**
- struct **_CpaInstanceInfo**
- struct **_CpaPhysicalInstanceId**
- struct **_CpaInstanceInfo2**

3.3 Defines

- #define **CPA_INSTANCE_HANDLE_SINGLE**
- #define **CPA_DP_BUFLIST**
- #define **CPA_STATUS_SUCCESS**
- #define **CPA_STATUS_FAIL**
- #define **CPA_STATUS_RETRY**
- #define **CPA_STATUS_RESOURCE**
- #define **CPA_STATUS_INVALID_PARAM**
- #define **CPA_STATUS_FATAL**
- #define **CPA_STATUS_UNSUPPORTED**
- #define **CPA_STATUS_RESTARTING**
- #define **CPA_STATUS_MAX_STR_LENGTH_IN_BYTES**
- #define **CPA_STATUS_STR_SUCCESS**
- #define **CPA_STATUS_STR_FAIL**
- #define **CPA_STATUS_STR_RETRY**
- #define **CPA_STATUS_STR_RESOURCE**
- #define **CPA_STATUS_STR_INVALID_PARAM**
- #define **CPA_STATUS_STR_FATAL**
- #define **CPA_STATUS_STR_UNSUPPORTED**
- #define **CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES**
- #define **CPA_INSTANCE_MAX_ID_SIZE_IN_BYTES**
- #define **CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES**

3.4 Typedefs

- typedef void * **CpaInstanceHandle**
- typedef **Cpa64U CpaPhysicalAddr**
- typedef **CpaPhysicalAddr**(* **CpaVirtualToPhysical**)(void *pVirtualAddr)
- typedef **_CpaFlatBuffer CpaFlatBuffer**
- typedef **_CpaBufferList CpaBufferList**
- typedef **_CpaPhysFlatBuffer CpaPhysFlatBuffer**
- typedef **_CpaPhysBufferList CpaPhysBufferList**
- typedef **Cpa32S CpaStatus**
- typedef enum **_CpaInstanceType CPA_DEPRECATED**
- typedef enum **_CpaAccelerationServiceType CpaAccelerationServiceType**
- typedef enum **_CpaInstanceState CPA_DEPRECATED**
- typedef enum **_CpaOperationalState CpaOperationalState**
- typedef **_CpaInstanceInfo CPA_DEPRECATED**
- typedef **_CpaPhysicalInstanceId CpaPhysicalInstanceId**
- typedef **_CpaInstanceInfo2 CpaInstanceInfo2**
- typedef enum **_CpaInstanceEvent CpaInstanceEvent**

3.5 Enumerations

- enum **_CpaInstanceType** {
  CPA_INSTANCE_TYPE_CRYPTO,
  CPA_INSTANCE_TYPE_DATA_COMPRESSION,
  CPA_INSTANCE_TYPE_RAID,
  CPA_INSTANCE_TYPE_XML,
  CPA_INSTANCE_TYPE_REGEX
  }
- enum **_CpaAccelerationServiceType** {
  CPA_ACC_SVC_TYPE_CRYPTO,
  CPA_ACC_SVC_TYPE_DATA_COMPRESSION,
  CPA_ACC_SVC_TYPE_PATTERN_MATCH,
  CPA_ACC_SVC_TYPE_RAID,
  CPA_ACC_SVC_TYPE_XML,
  CPA_ACC_SVC_TYPE_VIDEO_ANALYTICS
  }
- enum **_CpaInstanceState** {
  CPA_INSTANCE_STATE_INITIALISED,
  CPA_INSTANCE_STATE_SHUTDOWN
  }
- enum **_CpaOperationalState** {
  CPA_OPER_STATE_DOWN,
  CPA_OPER_STATE_UP
  }
- enum **_CpaInstanceEvent** {
  CPA_INSTANCE_EVENT_RESTARTING,
  CPA_INSTANCE_EVENT_RESTARTED,
  CPA_INSTANCE_EVENT_FATAL_ERROR
  }

3.6 Data Structure Documentation

3.6.1 _CpaFlatBuffer Struct Reference

3.6.1.1 Detailed Description

File: cpa.h

Flat buffer structure containing a pointer and length member.

A flat buffer structure. The data pointer, pData, is a virtual address. An API instance may require the actual data to be in contiguous physical memory as determined by **CpaInstanceInfo2**.

3.6.1.2 Data Fields

- **Cpa32U dataLenInBytes**
- **Cpa8U * pData**

3.6.1.3 Field Documentation

Cpa32U _CpaFlatBuffer::dataLenInBytes

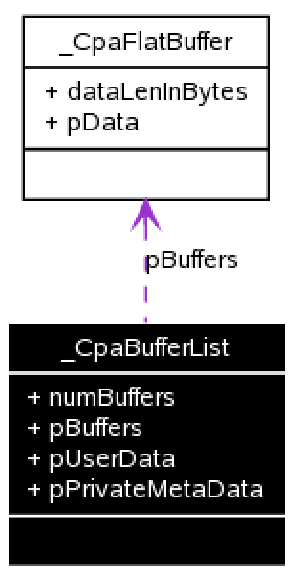
Data length specified in bytes. When used as an input parameter to a function, the length specifies the current length of the buffer. When used as an output parameter to a function, the length passed in specifies the maximum length of the buffer on return (i.e. the allocated length). The implementation will not write past this length. On return, the length is always unchanged.

Cpa8U* _CpaFlatBuffer::pData

The data pointer is a virtual address, however the actual data pointed to is required to be in contiguous physical memory unless the field requiresPhysicallyContiguousMemory in CpaInstanceInfo2 is false.

3.6.2 _CpaBufferList Struct Reference

Collaboration diagram for _CpaBufferList:



3.6.2.1 Detailed Description

File: cpa.h

Scatter/Gather buffer list containing an array of flat buffers.

A scatter/gather buffer list structure. This buffer structure is typically used to represent a region of memory which is not physically contiguous, by describing it as a collection of buffers, each of which is physically contiguous.

Note:

The memory for the pPrivateMetaData member must be allocated by the client as physically contiguous memory. When allocating memory for pPrivateMetaData, a call to the corresponding BufferListGetMetaSize function (e.g. cpaCyBufferListGetMetaSize) MUST be made to determine the size of the Meta Data Buffer. The returned size (in bytes) may then be passed in a memory allocation routine to allocate the pPrivateMetaData memory.

3.6.2.2 Data Fields

- **Cpa32U numBuffers**
- **CpaFlatBuffer** * **pBuffers**
- void * **pPrivateMetaData**

3.6.2.3 Field Documentation

Cpa32U _CpaBufferList::numBuffers

Number of buffers in the list

CpaFlatBuffer* **_CpaBufferList::pBuffers**

Pointer to an unbounded array containing the number of CpaFlatBuffers defined by numBuffers

void* **_CpaBufferList::pUserData**

This is an opaque field that is not read or modified internally.

void* **_CpaBufferList::pPrivateMetaData**

Private representation of this buffer list. The memory for this buffer needs to be allocated by the client as contiguous data. The amount of memory required is returned with a call to the corresponding BufferListGetMetaSize function. If that function returns a size of zero then no memory needs to be allocated, and this parameter can be NULL.

3.6.3 _CpaPhysFlatBuffer Struct Reference

3.6.3.1 Detailed Description

File: cpa.h

Flat buffer structure with physical address.

Functions taking this structure do not need to do any virtual to physical address translation before writing the buffer to hardware.

3.6.3.2 Data Fields

- **Cpa32U dataLenInBytes**
- **Cpa32U reserved**
- **CpaPhysicalAddr bufferPhysAddr**

3.6.3.3 Field Documentation

Cpa32U _CpaPhysFlatBuffer::dataLenInBytes

Data length specified in bytes. When used as an input parameter to a function, the length specifies the current length of the buffer. When used as an output parameter to a function, the length passed in specifies the maximum length of the buffer on return (i.e. the allocated length). The implementation will not write past this length. On return, the length is always unchanged.

Cpa32U _CpaPhysFlatBuffer::reserved

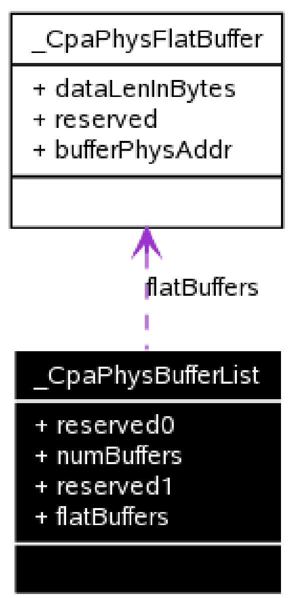
Reserved for alignment

CpaPhysicalAddr _CpaPhysFlatBuffer::bufferPhysAddr

The physical address at which the data resides. The data pointed to is required to be in contiguous physical memory.

3.6.4 _CpaPhysBufferList Struct Reference

Collaboration diagram for _CpaPhysBufferList:



3.6.4.1 Detailed Description

File: cpa.h

Scatter/gather list containing an array of flat buffers with physical addresses.

Similar to **CpaBufferList**, this buffer structure is typically used to represent a region of memory which is not physically contiguous, by describing it as a collection of buffers, each of which is physically contiguous. The difference is that, in this case, the individual "flat" buffers are represented using physical, rather than virtual, addresses.

3.6.4.2 Data Fields

- **Cpa64U reserved0**
- **Cpa32U numBuffers**
- **Cpa32U reserved1**
- **CpaPhysFlatBuffer flatBuffers** []

3.6.4.3 Field Documentation

Cpa64U _CpaPhysBufferList::reserved0

Reserved for internal usage

Cpa32U _CpaPhysBufferList::numBuffers

Number of buffers in the list

Cpa32U _CpaPhysBufferList::reserved1

Reserved for alignment

CpaPhysFlatBuffer _CpaPhysBufferList::flatBuffers[]

Array of flat buffer structures, of size numBuffers

3.6.5 _CpaInstanceInfo Struct Reference

3.6.5.1 Detailed Description

File: cpa.h

Instance Info Structure

Deprecated:

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by CpaInstanceInfo2.

Structure that contains the information to describe the instance.

3.6.5.2 Data Fields

- enum **_CpaInstanceType type**
- enum **_CpaInstanceState state**
- **Cpa8U name** [CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES]
- **Cpa8U version** [CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES]

3.6.5.3 Field Documentation

enum **_CpaInstanceType _CpaInstanceInfo::type**

Type definition for this instance.

enum **_CpaInstanceState _CpaInstanceInfo::state**

Operational state of the instance.

Cpa8U _CpaInstanceInfo::name[CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES]

Simple text string identifier for the instance.

Cpa8U _CpaInstanceInfo::version[CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES]

Version string. There may be multiple versions of the same type of instance accessible through a particular library.

3.6.6 _CpaPhysicalInstanceId Struct Reference

3.6.6.1 Detailed Description

File: cpa.h

Physical Instance ID

Identifies the physical instance of an accelerator execution engine.

Accelerators grouped into "packages". Each accelerator can in turn contain one or more execution engines. Implementations of this API will define the packageId, acceleratorId, executionEngineId and busAddress as appropriate for the implementation. For example, for hardware-based accelerators, the packageId might identify the chip, which might contain multiple accelerators, each of which might contain multiple execution engines. The combination of packageId, acceleratorId and executionEngineId uniquely identifies the instance.

Hardware based accelerators implementing this API may also provide information on the location of the accelerator in the busAddress field. This field will be defined as appropriate for the implementation. For example, for PCIe attached accelerators, the busAddress may contain the PCIe bus, device and function number of the accelerators.

3.6.6.2 Data Fields

- **Cpa16U packageId**
- **Cpa16U acceleratorId**
- **Cpa16U executionEngineId**
- **Cpa16U busAddress**
- **Cpa32U kptAcHandle**

3.6.6.3 Field Documentation

Cpa16U _CpaPhysicalInstanceId::packageId

Identifies the package within which the accelerator is contained.

Cpa16U _CpaPhysicalInstanceId::acceleratorId

Identifies the specific accelerator within the package.

Cpa16U _CpaPhysicalInstanceId::executionEngineId

Identifies the specific execution engine within the accelerator.

Cpa16U _CpaPhysicalInstanceId::busAddress

Identifies the bus address associated with the accelerator execution engine.

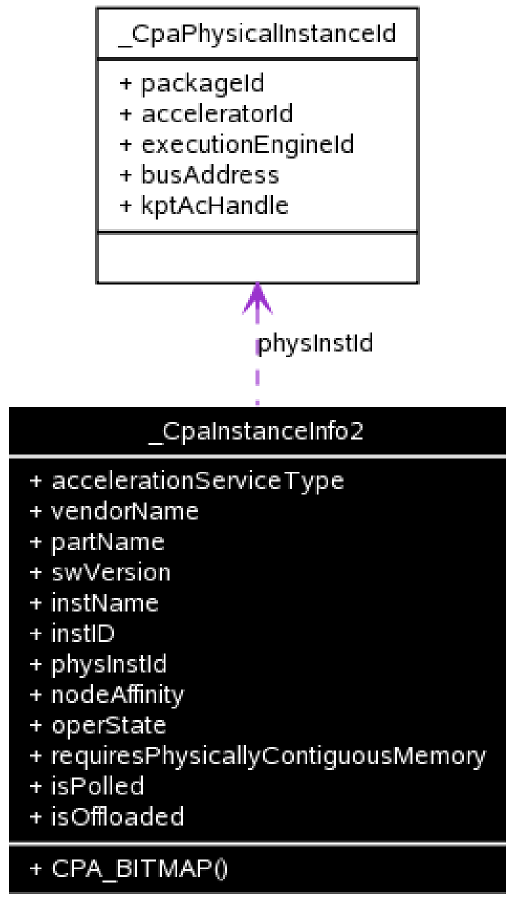
Cpa32U _CpaPhysicalInstanceId::kptAcHandle

3.6.6 _CpaPhysicalInstanceId Struct Reference

Identifies the achandle of the accelerator.

3.6.7 _CpaInstanceInfo2 Struct Reference

Collaboration diagram for _CpaInstanceInfo2:



3.6.7.1 Detailed Description

File: cpa.h

Instance Info Structure, version 2

Structure that contains the information to describe the instance.

3.6.7.2 Public Member Functions

- **CPA_BITMAP** (coreAffinity, CPA_MAX_CORES)

3.6.7.3 Data Fields

- **CpaAccelerationServiceType accelerationServiceType**
- **Cpa8U vendorName** [CPA_INST_VENDOR_NAME_SIZE]
- **Cpa8U partName** [CPA_INST_PART_NAME_SIZE]
- **Cpa8U swVersion** [CPA_INST_SW_VERSION_SIZE]
- **Cpa8U instName** [CPA_INST_NAME_SIZE]

3.6.7 _CpaInstanceInfo2 Struct Reference

- **Cpa8U instID** [CPA_INST_ID_SIZE]
- **CpaPhysicalInstanceId physInstId**
- **Cpa32U nodeAffinity**
- **CpaOperationalState operState**
- **CpaBoolean requiresPhysicallyContiguousMemory**
- **CpaBoolean isPolled**
- **CpaBoolean isOffloaded**

3.6.7.4 Member Function Documentation

```
_CpaInstanceInfo2::CPA_BITMAP( coreAffinity   ,
                               CPA_MAX_CORES
                             )
```

A bitmap identifying the core or cores to which the instance is affinitized in an SMP operating system.

The term core here is used to mean a "logical" core - for example, in a dual-processor, quad-core system with hyperthreading (two threads per core), there would be 16 such cores (2 processors x 4 cores/processor x 2 threads/core). The numbering of these cores and the corresponding bit positions is OS-specific. Note that Linux refers to this as "processor affinity" or "CPU affinity", and refers to the bitmap as a "cpumask".

The term "affinity" is used to mean that this is the core on which the callback function will be invoked when using the asynchronous mode of the API. In a hardware-based implementation of the API, this might be the core to which the interrupt is affinitized. In a software-based implementation, this might be the core to which the process running the algorithm is affinitized. Where there is no affinity, the bitmap can be set to all zeroes.

This bitmap should be manipulated using the macros **CPA_BITMAP_BIT_SET**, **CPA_BITMAP_BIT_CLEAR** and **CPA_BITMAP_BIT_TEST**.

3.6.7.5 Field Documentation

CpaAccelerationServiceType _CpaInstanceInfo2::accelerationServiceType

Type of service provided by this instance.

Cpa8U _CpaInstanceInfo2::vendorName[CPA_INST_VENDOR_NAME_SIZE]

String identifying the vendor of the accelerator.

Cpa8U _CpaInstanceInfo2::partName[CPA_INST_PART_NAME_SIZE]

String identifying the part (name and/or number).

Cpa8U _CpaInstanceInfo2::swVersion[CPA_INST_SW_VERSION_SIZE]

String identifying the version of the software associated with the instance. For hardware-based implementations of the API, this should be the driver version. For software-based implementations of the API, this should be the version of the library.

Note that this should NOT be used to store the version of the API, nor should it be used to report the hardware revision (which can be captured as part of the **partName**, if required).

Cpa8U _CpaInstanceInfo2::instName[CPA_INST_NAME_SIZE]

String identifying the name of the instance.

Cpa8U _CpaInstanceInfo2::instID[CPA_INST_ID_SIZE]

String containing a unique identifier for the instance

CpaPhysicalInstanceId _CpaInstanceInfo2::physInstId

Identifies the "physical instance" of the accelerator.

Cpa32U _CpaInstanceInfo2::nodeAffinity

Identifies the processor complex, or node, to which the accelerator is physically connected, to help identify locality in NUMA systems.

The values taken by this attribute will typically be in the range 0..n-1, where n is the number of nodes (processor complexes) in the system. For example, in a dual-processor configuration, n=2. The precise values and their interpretation are OS-specific.

CpaOperationalState _CpaInstanceInfo2::operState

Operational state of the instance.

CpaBoolean _CpaInstanceInfo2::requiresPhysicallyContiguousMemory

Specifies whether the data pointed to by flat buffers (**CpaFlatBuffer::pData**) supplied to this instance must be in physically contiguous memory.

CpaBoolean _CpaInstanceInfo2::isPolled

Specifies whether the instance must be polled, or is event driven. For hardware accelerators, the alternative to polling would be interrupts.

CpaBoolean _CpaInstanceInfo2::isOffloaded

Identifies whether the instance uses hardware offload, or is a software-only implementation.

3.7 Define Documentation

#define CPA_INSTANCE_HANDLE_SINGLE

File: cpa.h

Default instantiation handle value where there is only a single instance

Used as an instance handle value where only one instance exists.

#define CPA_DP_BUFLIST

File: cpa.h

Special value which can be taken by length fields on some of the "data plane" APIs to indicate that the buffer in question is of type CpaPhysBufferList, rather than simply an array of bytes.

#define CPA_STATUS_SUCCESS

Success status value.

#define CPA_STATUS_FAIL

Fail status value.

#define CPA_STATUS_RETRY

Retry status value.

#define CPA_STATUS_RESOURCE

The resource that has been requested is unavailable. Refer to relevant sections of the API for specifics on what the suggested course of action is.

#define CPA_STATUS_INVALID_PARAM

Invalid parameter has been passed in.

#define CPA_STATUS_FATAL

A serious error has occurred. Recommended course of action is to shutdown and restart the component.

#define CPA_STATUS_UNSUPPORTED

The function is not supported, at least not with the specific parameters supplied. This may be because a particular capability is not supported by the current implementation.

#define CPA_STATUS_RESTARTING

The API implementation is restarting. This may be reported if, for example, a hardware implementation is undergoing a reset. Recommended course of action is to retry the request.

#define CPA_STATUS_MAX_STR_LENGTH_IN_BYTES

File: cpa.h

API status string type definition

This type definition is used for the generic status text strings provided by cpaXxGetStatusText API functions. Common values are defined, for example see **CPA_STATUS_STR_SUCCESS**, **CPA_STATUS_FAIL**, etc., as well as the maximum size **CPA_STATUS_MAX_STR_LENGTH_IN_BYTES**.

Maximum length of the Overall Status String (including generic and specific strings returned by calls to cpaXxGetStatusText)

#define CPA_STATUS_STR_SUCCESS

Status string for **CPA_STATUS_SUCCESS**.

#define CPA_STATUS_STR_FAIL

Status string for **CPA_STATUS_FAIL**.

#define CPA_STATUS_STR_RETRY

Status string for **CPA_STATUS_RETRY**.

#define CPA_STATUS_STR_RESOURCE

Status string for **CPA_STATUS_RESOURCE**.

#define CPA_STATUS_STR_INVALID_PARAM

Status string for **CPA_STATUS_INVALID_PARAM**.

#define CPA_STATUS_STR_FATAL

Status string for **CPA_STATUS_FATAL**.

#define CPA_STATUS_STR_UNSUPPORTED

Status string for **CPA_STATUS_UNSUPPORTED**.

#define CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES
Maximum instance info name string length in bytes

#define CPA_INSTANCE_MAX_ID_SIZE_IN_BYTES
Maximum instance info id string length in bytes

#define CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES
Maximum instance info version string length in bytes

3.8 Typedef Documentation

typedef void* **CpaInstanceHandle**

File: cpa.h

Instance handle type.

Handle used to uniquely identify an instance.

Note:
Where only a single instantiation exists this field may be set to **CPA_INSTANCE_HANDLE_SINGLE**.

typedef **Cpa64U** **CpaPhysicalAddr**

File: cpa.h

Physical memory address.

Type for physical memory addresses.

typedef **CpaPhysicalAddr**(* **CpaVirtualToPhysical**)(void *pVirtualAddr)

File: cpa.h

Virtual to physical address conversion routine.

This function is used to convert virtual addresses to physical addresses.

Context:
The function shall not be called in an interrupt context.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:
`[in]` *pVirtualAddr* Virtual address to be converted.

Returns:
Returns the corresponding physical address. On error, the value NULL is returned.

Postcondition:
None

See also:
None

typedef struct **_CpaFlatBuffer CpaFlatBuffer**

File: cpa.h

Flat buffer structure containing a pointer and length member.

A flat buffer structure. The data pointer, pData, is a virtual address. An API instance may require the actual data to be in contiguous physical memory as determined by **CpaInstanceInfo2**.

typedef struct **_CpaBufferList CpaBufferList**

File: cpa.h

Scatter/Gather buffer list containing an array of flat buffers.

A scatter/gather buffer list structure. This buffer structure is typically used to represent a region of memory which is not physically contiguous, by describing it as a collection of buffers, each of which is physically contiguous.

Note:
The memory for the pPrivateMetaData member must be allocated by the client as physically contiguous memory. When allocating memory for pPrivateMetaData, a call to the corresponding BufferListGetMetaSize function (e.g. cpaCyBufferListGetMetaSize) MUST be made to determine the size of the Meta Data Buffer. The returned size (in bytes) may then be passed in a memory allocation routine to allocate the pPrivateMetaData memory.

typedef struct **_CpaPhysFlatBuffer CpaPhysFlatBuffer**

File: cpa.h

Flat buffer structure with physical address.

Functions taking this structure do not need to do any virtual to physical address translation before writing the buffer to hardware.

typedef struct **_CpaPhysBufferList CpaPhysBufferList**

File: cpa.h

Scatter/gather list containing an array of flat buffers with physical addresses.

Similar to **CpaBufferList**, this buffer structure is typically used to represent a region of memory which is not physically contiguous, by describing it as a collection of buffers, each of which is physically contiguous. The difference is that, in this case, the individual "flat" buffers are represented using physical, rather than virtual, addresses.

typedef **Cpa32S CpaStatus**

File: cpa.h

API status value type definition

This type definition is used for the return values used in all the API functions. Common values are defined, for example see **CPA_STATUS_SUCCESS**, **CPA_STATUS_FAIL**, etc.

typedef enum **_CpaInstanceType CPA_DEPRECATED**

File: cpa.h

Instance Types

Deprecated:
As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaAccelerationServiceType**.

Enumeration of the different instance types.

typedef enum **_CpaAccelerationServiceType CpaAccelerationServiceType**

File: cpa.h

Service Type

Enumeration of the different service types.

typedef enum **_CpaInstanceState CPA_DEPRECATED**

File: cpa.h

Instance State

Deprecated:
As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaOperationalState**.

Enumeration of the different instance states that are possible.

typedef enum **_CpaOperationalState CpaOperationalState**

File: cpa.h

Instance operational state

Enumeration of the different operational states that are possible.

typedef struct **_CpaInstanceInfo CPA_DEPRECATED**

File: cpa.h

Instance Info Structure

Deprecated:
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by CpaInstanceInfo2.

Structure that contains the information to describe the instance.

typedef struct **_CpaPhysicalInstanceId CpaPhysicalInstanceId**

File: cpa.h

Physical Instance ID

Identifies the physical instance of an accelerator execution engine.

Accelerators grouped into "packages". Each accelerator can in turn contain one or more execution engines. Implementations of this API will define the packageId, acceleratorId, executionEngineId and busAddress as appropriate for the implementation. For example, for hardware-based accelerators, the packageId might identify the chip, which might contain multiple accelerators, each of which might contain multiple execution engines. The combination of packageId, acceleratorId and executionEngineId uniquely identifies the instance.

Hardware based accelerators implementing this API may also provide information on the location of the accelerator in the busAddress field. This field will be defined as appropriate for the implementation. For example, for PCIe attached accelerators, the busAddress may contain the PCIe bus, device and function number of the accelerators.

typedef struct **_CpaInstanceInfo2 CpaInstanceInfo2**

File: cpa.h

Instance Info Structure, version 2

Structure that contains the information to describe the instance.

typedef enum **_CpaInstanceEvent CpaInstanceEvent**

File: cpa.h

Instance Events

Enumeration of the different events that will cause the registered Instance notification callback function to be invoked.

3.9 Enumeration Type Documentation

enum **_CpaInstanceType**

File: cpa.h

Instance Types

Deprecated:
As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaAccelerationServiceType**.

Enumeration of the different instance types.

Enumerator:

CPA_INSTANCE_TYPE_CRYPTO	Cryptographic instance type
CPA_INSTANCE_TYPE_DATA_COMPRESSION	Data compression instance type
CPA_INSTANCE_TYPE_RAID	RAID instance type
CPA_INSTANCE_TYPE_XML	XML instance type
CPA_INSTANCE_TYPE_REGEX	Regular Expression instance type

enum **_CpaAccelerationServiceType**

File: cpa.h

Service Type

Enumeration of the different service types.

Enumerator:

CPA_ACC_SVC_TYPE_CRYPTO	Cryptography
CPA_ACC_SVC_TYPE_DATA_COMPRESSION	Data Compression
CPA_ACC_SVC_TYPE_PATTERN_MATCH	Pattern Match
CPA_ACC_SVC_TYPE_RAID	RAID
CPA_ACC_SVC_TYPE_XML	XML
CPA_ACC_SVC_TYPE_VIDEO_ANALYTICS	Video Analytics

enum **_CpaInstanceState**

File: cpa.h

Instance State

Deprecated:
As of v1.3 of the Crypto API, this enum has been deprecated, replaced by **CpaOperationalState**.

Enumeration of the different instance states that are possible.

Enumerator:

CPA_INSTANCE_STATE_INITIALISED	Instance is in the initialized state and ready for use.

CPA_INSTANCE_STATE_SHUTDOWN	Instance is in the shutdown state and not available for use.

enum **_CpaOperationalState**

File: cpa.h

Instance operational state

Enumeration of the different operational states that are possible.

Enumerator:

CPA_OPER_STATE_DOWN	Instance is not available for use. May not yet be initialized, or stopped.
CPA_OPER_STATE_UP	Instance is available for use. Has been initialized and started.

enum **_CpaInstanceEvent**

File: cpa.h

Instance Events

Enumeration of the different events that will cause the registered Instance notification callback function to be invoked.

Enumerator:

CPA_INSTANCE_EVENT_RESTARTING	Event type that triggers the registered instance notification callback function when and instance is restarting. The reason why an instance is restarting is implementation specific. For example a hardware implementation may send this event if the hardware device is about to be reset.
CPA_INSTANCE_EVENT_RESTARTED	Event type that triggers the registered instance notification callback function when and instance has restarted. The reason why an instance has restarted is implementation specific. For example a hardware implementation may send this event after the hardware device has been reset.
CPA_INSTANCE_EVENT_FATAL_ERROR	Event type that triggers the registered instance notification callback function when an error has been detected that requires the device to be reset. This event will be sent by all instances using the device, both on the host and guests.

4 CPA Type Definition

[CPA API]

Collaboration diagram for CPA Type Definition:

4.1 Detailed Description

File: cpa_types.h

This is the CPA Type Definitions.

4.2 Defines

- #define **NULL**
- #define **TRUE**
- #define **FALSE**
- #define **CPA_BITMAP**(name, sizeInBits)
- #define **CPA_BITMAP_BIT_TEST**(bitmask, bit)
- #define **CPA_BITMAP_BIT_SET**(bitmask, bit)
- #define **CPA_BITMAP_BIT_CLEAR**(bitmask, bit)
- #define **CPA_DEPRECATED**

4.3 Typedefs

- typedef uint8_t **Cpa8U**
- typedef int8_t **Cpa8S**
- typedef uint16_t **Cpa16U**
- typedef int16_t **Cpa16S**
- typedef uint32_t **Cpa32U**
- typedef int32_t **Cpa32S**
- typedef uint64_t **Cpa64U**
- typedef int64_t **Cpa64S**
- typedef enum **_CpaBoolean CpaBoolean**

4.4 Enumerations

- enum **_CpaBoolean** {
  CPA_FALSE,
  CPA_TRUE
  }

4.5 Define Documentation

#define NULL

File: cpa_types.h

NULL definition.

#define TRUE

File: cpa_types.h

True value definition.

#define FALSE

File: cpa_types.h

False value definition.

#define CPA_BITMAP (name,
sizeInBits)

File: cpa_types.h

Declare a bitmap of specified size (in bits).

This macro is used to declare a bitmap of arbitrary size.

To test whether a bit in the bitmap is set, use **CPA_BITMAP_BIT_TEST**.

While most uses of bitmaps on the API are read-only, macros are also provided to set (see **CPA_BITMAP_BIT_SET**) and clear (see **CPA_BITMAP_BIT_CLEAR**) bits in the bitmap.

#define CPA_BITMAP_BIT_TEST (bitmask,
bit)

Test a specified bit in the specified bitmap. The bitmap may have been declared using **CPA_BITMAP**. Returns a Boolean (true if the bit is set, false otherwise).

#define CPA_BITMAP_BIT_SET (bitmask,
bit)

File: cpa_types.h

Set a specified bit in the specified bitmap. The bitmap may have been declared using **CPA_BITMAP**.

#define CPA_BITMAP_BIT_CLEAR (bitmask,
bit)

Clear a specified bit in the specified bitmap. The bitmap may have been declared using **CPA_BITMAP**.

#define **CPA_DEPRECATED**

Declare a function or type and mark it as deprecated so that usages get flagged with a warning.

4.6 Typedef Documentation

typedef uint8_t **Cpa8U**

File: cpa_types.h

Unsigned byte base type.

typedef int8_t **Cpa8S**

File: cpa_types.h

Signed byte base type.

typedef uint16_t **Cpa16U**

File: cpa_types.h

Unsigned double-byte base type.

typedef int16_t **Cpa16S**

File: cpa_types.h

Signed double-byte base type.

typedef uint32_t **Cpa32U**

File: cpa_types.h

Unsigned quad-byte base type.

typedef int32_t **Cpa32S**

File: cpa_types.h

Signed quad-byte base type.

typedef uint64_t **Cpa64U**

File: cpa_types.h

Unsigned double-quad-byte base type.

typedef int64_t **Cpa64S**

File: cpa_types.h

Signed double-quad-byte base type.

typedef enum **_CpaBoolean CpaBoolean**

File: cpa_types.h

Boolean type.

Functions in this API use this type for Boolean variables that take true or false values.

4.7 Enumeration Type Documentation

enum **_CpaBoolean**

File: cpa_types.h

Boolean type.

Functions in this API use this type for Boolean variables that take true or false values.

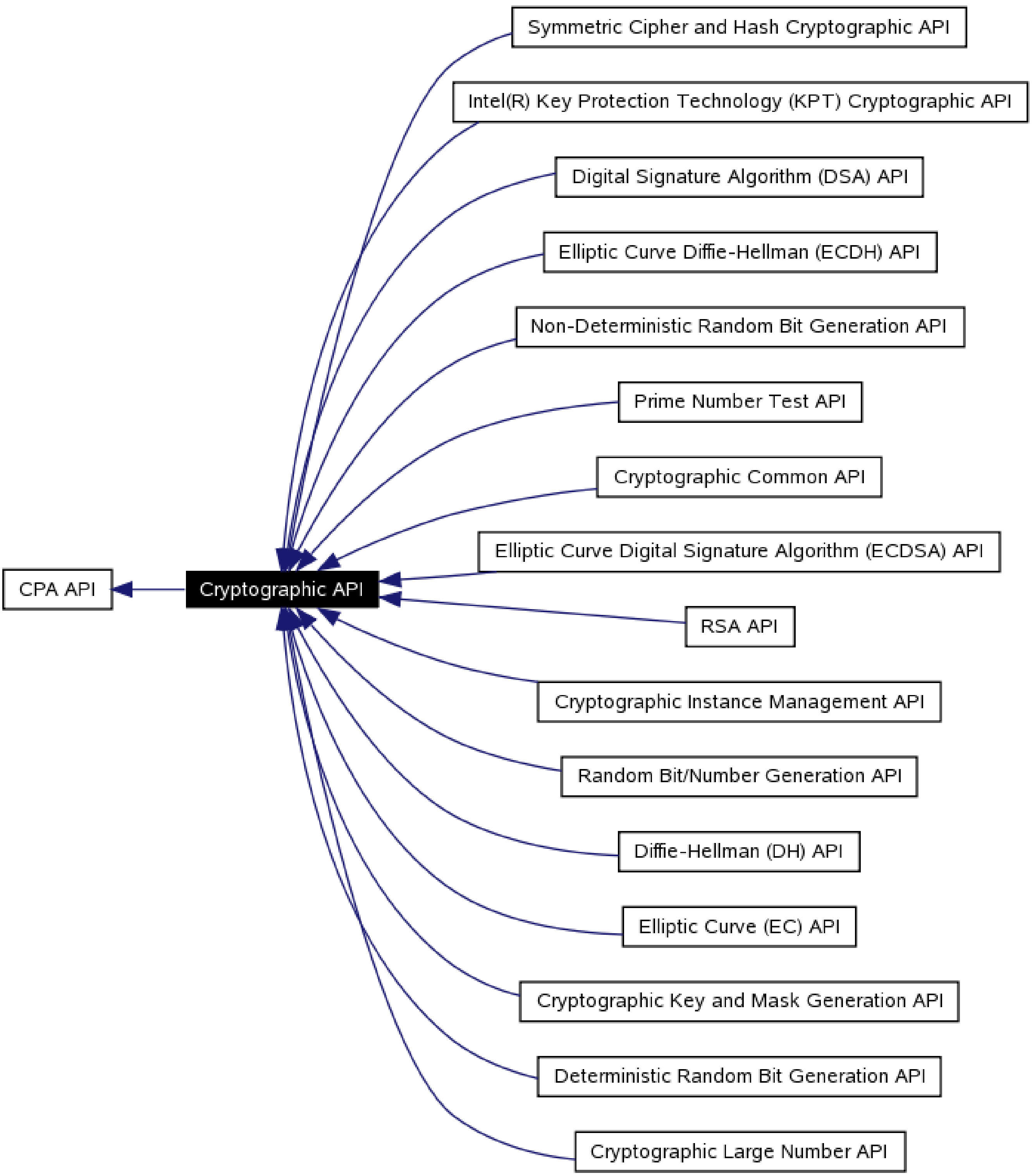
Enumerator:

CPA_FALSE	False value
CPA_TRUE	True value

5 Cryptographic API

[CPA API]

Collaboration diagram for Cryptographic API:



5.1 Detailed Description

File: cpa_cy_common.h

These functions specify the Cryptographic API.

5.2 Modules

- **Cryptographic Common API**
- **Cryptographic Instance Management API**
- **Symmetric Cipher and Hash Cryptographic API**
- **Cryptographic Key and Mask Generation API**
- **RSA API**
- **Diffie-Hellman (DH) API**
- **Digital Signature Algorithm (DSA) API**
- **Elliptic Curve (EC) API**
- **Elliptic Curve Diffie-Hellman (ECDH) API**
- **Elliptic Curve Digital Signature Algorithm (ECDSA) API**
- **Cryptographic Large Number API**
- **Prime Number Test API**
- **Deterministic Random Bit Generation API**
- **Non-Deterministic Random Bit Generation API**
- **Random Bit/Number Generation API**
- **Intel(R) Key Protection Technology (KPT) Cryptographic API**

6 Cryptographic Common API

[Cryptographic API]

Collaboration diagram for Cryptographic Common API:

6.1 Detailed Description

File: cpa_cy_common.h

This file specifies items which are common for both the asymmetric (public key cryptography) and the symmetric operations for the Cryptographic API.

6.2 Typedefs

- typedef enum **_CpaCyPriority CpaCyPriority**
- typedef void(* **CpaCyGenericCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData)
- typedef void(* **CpaCyGenFlatBufCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpdata, **CpaFlatBuffer** *pOut)
- typedef void(* **CpaCyInstanceNotificationCbFunc**)(const **CpaInstanceHandle** instanceHandle, void *pCallbackTag, const **CpaInstanceEvent** instanceEvent)

6.3 Enumerations

- enum **_CpaCyPriority** {
  CPA_CY_PRIORITY_NORMAL,
  CPA_CY_PRIORITY_HIGH
  }

6.4 Functions

- **CpaStatus cpaCyBufferListGetMetaSize** (const **CpaInstanceHandle** instanceHandle, **Cpa32U** numBuffers, **Cpa32U** *pSizeInBytes)
- **CpaStatus cpaCyGetStatusText** (const **CpaInstanceHandle** instanceHandle, **CpaStatus** errStatus, **Cpa8S** *pStatusText)
- **CpaStatus cpaCyGetNumInstances** (**Cpa16U** *pNumInstances)
- **CpaStatus cpaCyGetInstances** (**Cpa16U** numInstances, **CpaInstanceHandle** *cyInstances)
- **CpaStatus CPA_DEPRECATED cpaCyInstanceGetInfo** (const **CpaInstanceHandle** instanceHandle, struct **_CpaInstanceInfo** *pInstanceInfo)
- **CpaStatus cpaCyInstanceGetInfo2** (const **CpaInstanceHandle** instanceHandle, **CpaInstanceInfo2** *pInstanceInfo2)
- **CpaStatus cpaCyInstanceSetNotificationCb** (const **CpaInstanceHandle** instanceHandle, const **CpaCyInstanceNotificationCbFunc** pInstanceNotificationCb, void *pCallbackTag)

6.5 Typedef Documentation

typedef enum **_CpaCyPriority CpaCyPriority**

File: cpa_cy_common.h

Request priority

Enumeration of priority of the request to be given to the API. Currently two levels - HIGH and NORMAL are supported. HIGH priority requests will be prioritized on a "best-effort" basis over requests that are marked with a NORMAL priority.

typedef void(* **CpaCyGenericCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData)

File: cpa_cy_common.h

Definition of the crypto generic callback function

This data structure specifies the prototype for a generic callback function

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*pCallbackTag*	Opaque value provided by user while making individual function call.
[in]	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
[in]	*pOpData*	Opaque Pointer to the operation data that was submitted in the request

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:

cpaCyKeyGenSsl()

typedef void(* **CpaCyGenFlatBufCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpdata, **CpaFlatBuffer** *pOut)

File: cpa_cy_common.h

Definition of generic callback function with an additional output CpaFlatBuffer parameter.

This data structure specifies the prototype for a generic callback function which provides an output buffer (of type CpaFlatBuffer).

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*pCallbackTag*	Opaque value provided by user while making individual function call.
`[in]`	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
`[in]`	*pOpData*	Opaque Pointer to the operation data that was submitted in the request
`[in]`	*pOut*	Pointer to the output buffer provided in the request invoking this callback.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
None

typedef void(* **CpaCyInstanceNotificationCbFunc**)(const **CpaInstanceHandle** instanceHandle, void *pCallbackTag, const **CpaInstanceEvent** instanceEvent)

File: cpa_cy_common.h

Callback function for instance notification support.

This is the prototype for the instance notification callback function. The callback function is passed in as a parameter to the **cpaCyInstanceSetNotificationCb** function.

Context:
This function will be executed in a context that requires that sleeping MUST NOT be permitted.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCallbackTag*	Opaque value provided by user while making individual function calls.
[in]	*instanceEvent*	The event that will trigger this function to get invoked.

Return values:
None

Precondition:
Component has been initialized and the notification function has been set via the cpaCyInstanceSetNotificationCb function.

Postcondition:
None

Note:
None

See also:
cpaCyInstanceSetNotificationCb(),

6.6 Enumeration Type Documentation

enum **_CpaCyPriority**

File: cpa_cy_common.h

Request priority

Enumeration of priority of the request to be given to the API. Currently two levels - HIGH and NORMAL are supported. HIGH priority requests will be prioritized on a "best-effort" basis over requests that are marked with a NORMAL priority.

Enumerator:

CPA_CY_PRIORITY_NORMAL	Normal priority
CPA_CY_PRIORITY_HIGH	High priority

6.7 Function Documentation

CpaStatus cpaCyBufferListGetMetaSize (const **CpaInstanceHandle** *instanceHandle*,
Cpa32U *numBuffers*,
Cpa32U * *pSizeInBytes*
)

File: cpa_cy_common.h

Function to return the size of the memory which must be allocated for the pPrivateMetaData member of CpaBufferList.

This function is used obtain the size (in bytes) required to allocate a buffer descriptor for the pPrivateMetaData member in the CpaBufferList the structure. Should the function return zero then no meta data is required for the buffer list.

Context:
This function may be called from any context.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Handle to an instance of this API.
`[in]`	*numBuffers*	The number of pointers in the CpaBufferList. this is the maximum number of CpaFlatBuffers which may be contained in this CpaBufferList.
`[out]`	*pSizeInBytes*	Pointer to the size in bytes of memory to be allocated when the client wishes to allocate a cpaFlatBuffer

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
None.

Postcondition:
None

Note:
None

See also:
cpaCyGetInstances()

CpaStatus cpaCyGetStatusText (	const **CpaInstanceHandle**	*instanceHandle*,
	CpaStatus	*errStatus*,
	Cpa8S *	*pStatusText*
)		

File: cpa_cy_common.h

Function to return a string indicating the specific error that occurred for a particular instance.

When a function invocation on a particular instance returns an error, the client can invoke this function to query the instance for a null terminated string which describes the general error condition, and if available additional text on the specific error. The Client MUST allocate CPA_STATUS_MAX_STR_LENGTH_IN_BYTES bytes for the buffer string.

Context:
This function may be called from any context.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Handle to an instance of this API.
`[in]`	*errStatus*	The error condition that occurred
`[out]`	*pStatusText*	Pointer to the string buffer that will be updated with a null terminated status text string. The invoking application MUST allocate this buffer to be CPA_STATUS_MAX_STR_LENGTH_IN_BYTES.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed. Note, In this scenario it is INVALID to call this function a further time.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
None.

Postcondition:
None

Note:
None

See also:
CpaStatus

CpaStatus cpaCyGetNumInstances (**Cpa16U** * *pNumInstances*)

File: cpa_cy_common.h

Get the number of instances that are supported by the API implementation.

This function will get the number of instances that are supported by an implementation of the Cryptographic API. This number is then used to determine the size of the array that must be passed to **cpaCyGetInstances()**.

Context:
This function MUST NOT be called from an interrupt context as it MAY sleep.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:
`[out]` *pNumInstances* Pointer to where the number of instances will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
None

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated

See also:
cpaCyGetInstances

CpaStatus cpaCyGetInstances	(	**Cpa16U**	*numInstances*,
		CpaInstanceHandle *	*cyInstances*
	)		

File: cpa_cy_common.h

Get the handles to the instances that are supported by the API implementation.

This function will return handles to the instances that are supported by an implementation of the Cryptographic API. These instance handles can then be used as input parameters with other Cryptographic API functions.

This function will populate an array that has been allocated by the caller. The size of this API will have been determined by the **cpaCyGetNumInstances()** function.

Context:
This function MUST NOT be called from an interrupt context as it MAY sleep.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*numInstances*	Size of the array. If the value is not the same as the number of instances supported, then an error (**CPA_STATUS_INVALID_PARAM**) is returned.
`[in,out]`	*cyInstances*	Pointer to where the instance handles will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
None

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated

See also:
cpaCyGetNumInstances

CpaStatus CPA_DEPRECATED cpaCyInstanceGetInfo	(	const **CpaInstanceHandle**	*instanceHandle*,
		struct **_CpaInstanceInfo** *	*pInstanceInfo*
	)		

File: cpa_cy_common.h

Function to get information on a particular instance.

Deprecated:
As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyInstanceGetInfo2**.

This function will provide instance specific information through a CpaInstanceInfo structure.

Context:
This function may be called from any context.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Handle to an instance of this API to be initialized.
`[out]`	*pInstanceInfo*	Pointer to the memory location allocated by the client into which the CpaInstanceInfo structure will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The client has retrieved an instanceHandle from successive calls to **cpaCyGetNumInstances** and **cpaCyGetInstances**.

Postcondition:
None

Note:
None

See also:
cpaCyGetNumInstances, **cpaCyGetInstances**, CpaInstanceInfo

CpaStatus cpaCyInstanceGetInfo2 (const **CpaInstanceHandle** *instanceHandle*, **CpaInstanceInfo2** * *pInstanceInfo2*)

Function to get information on a particular instance.

This function will provide instance specific information through a **CpaInstanceInfo2** structure. Supersedes **cpaCyInstanceGetInfo**.

Context:
This function may be called from any context.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Handle to an instance of this API to be initialized.
`[out]`	*pInstanceInfo2*	Pointer to the memory location allocated by the client into which the CpaInstanceInfo2 structure will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The client has retrieved an instanceHandle from successive calls to **cpaCyGetNumInstances** and **cpaCyGetInstances**.

Postcondition:
None

Note:
None

See also:
cpaCyGetNumInstances, **cpaCyGetInstances**, CpaInstanceInfo

CpaStatus cpaCyInstanceSetNotificationCb	(const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyInstanceNotificationCbFunc**	*pInstanceNotificationCb*,
	void *	*pCallbackTag*
	)	

File: cpa_cy_common.h

Subscribe for instance notifications.

Clients of the CpaCy interface can subscribe for instance notifications by registering a **CpaCyInstanceNotificationCbFunc** function.

Context:
This function may be called from any context.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pInstanceNotificationCb*	Instance notification callback function pointer.
[in]	*pCallbackTag*	Opaque value provided by user while making individual function calls.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Instance has been initialized.

Postcondition:
None

Note:
None

See also:

CpaCyInstanceNotificationCbFunc

7 Cryptographic Instance Management API

[Cryptographic API]

Collaboration diagram for Cryptographic Instance Management API:

7.1 Detailed Description

File: cpa_cy_im.h

These functions specify the Instance Management API for available Cryptographic Instances. It is expected that these functions will only be called via a single system maintenance entity, rather than individual clients.

7.2 Data Structures

- struct **_CpaCyCapabilitiesInfo**

7.3 Typedefs

- typedef **_CpaCyCapabilitiesInfo CpaCyCapabilitiesInfo**

7.4 Functions

- **CpaStatus cpaCyStartInstance** (**CpaInstanceHandle** instanceHandle)
- **CpaStatus cpaCyStopInstance** (**CpaInstanceHandle** instanceHandle)
- **CpaStatus cpaCyQueryCapabilities** (const **CpaInstanceHandle** instanceHandle, **CpaCyCapabilitiesInfo** *pCapInfo)
- **CpaStatus cpaCySetAddressTranslation** (const **CpaInstanceHandle** instanceHandle, **CpaVirtualToPhysical** virtual2Physical)

7.5 Data Structure Documentation

7.5.1 _CpaCyCapabilitiesInfo Struct Reference

7.5.1.1 Detailed Description

File: cpa_cy_im.h

Cryptographic Capabilities Info

This structure contains the capabilities that vary across API implementations. This structure is used in conjunction with **cpaCyQueryCapabilities()** to determine the capabilities supported by a particular API implementation.

The client MUST allocate memory for this structure and any members that require memory. When the structure is passed into the function ownership of the memory passes to the function. Ownership of the memory returns to the client when the function returns.

7.5.1.2 Data Fields

- **CpaBoolean symSupported**
- **CpaBoolean symDpSupported**
- **CpaBoolean dhSupported**
- **CpaBoolean dsaSupported**
- **CpaBoolean rsaSupported**
- **CpaBoolean ecSupported**
- **CpaBoolean ecdhSupported**
- **CpaBoolean ecdsaSupported**
- **CpaBoolean keySupported**
- **CpaBoolean lnSupported**
- **CpaBoolean primeSupported**
- **CpaBoolean drbgSupported**
- **CpaBoolean nrbgSupported**
- **CpaBoolean randSupported**
- **CpaBoolean kptSupported**

7.5.1.3 Field Documentation

CpaBoolean _CpaCyCapabilitiesInfo::symSupported

CPA_TRUE if instance supports the symmetric cryptography API. See **Symmetric Cipher and Hash Cryptographic API**.

CpaBoolean _CpaCyCapabilitiesInfo::symDpSupported

CPA_TRUE if instance supports the symmetric cryptography data plane API. See **Symmetric cryptographic Data Plane API**.

CpaBoolean _CpaCyCapabilitiesInfo::dhSupported

CPA_TRUE if instance supports the Diffie Hellman API. See **Diffie-Hellman (DH) API**.

CpaBoolean _CpaCyCapabilitiesInfo::dsaSupported

CPA_TRUE if instance supports the DSA API. See **Digital Signature Algorithm (DSA) API**.

CpaBoolean _CpaCyCapabilitiesInfo::rsaSupported

CPA_TRUE if instance supports the RSA API. See **RSA API**.

CpaBoolean _CpaCyCapabilitiesInfo::ecSupported

CPA_TRUE if instance supports the Elliptic Curve API. See **Elliptic Curve (EC) API**.

CpaBoolean _CpaCyCapabilitiesInfo::ecdhSupported

CPA_TRUE if instance supports the Elliptic Curve Diffie Hellman API. See **Elliptic Curve Diffie-Hellman (ECDH) API**.

CpaBoolean _CpaCyCapabilitiesInfo::ecdsaSupported

CPA_TRUE if instance supports the Elliptic Curve DSA API. See **Elliptic Curve Digital Signature Algorithm (ECDSA) API**.

CpaBoolean _CpaCyCapabilitiesInfo::keySupported

CPA_TRUE if instance supports the Key Generation API. See **Cryptographic Key and Mask Generation API**.

CpaBoolean _CpaCyCapabilitiesInfo::lnSupported

CPA_TRUE if instance supports the Large Number API. See **Cryptographic Large Number API**.

CpaBoolean _CpaCyCapabilitiesInfo::primeSupported

CPA_TRUE if instance supports the prime number testing API. See **Prime Number Test API**.

CpaBoolean _CpaCyCapabilitiesInfo::drbgSupported

CPA_TRUE if instance supports the DRBG API. See **Deterministic Random Bit Generation API**.

CpaBoolean _CpaCyCapabilitiesInfo::nrbgSupported

CPA_TRUE if instance supports the NRBG API. See **Non-Deterministic Random Bit Generation API**.

CpaBoolean _CpaCyCapabilitiesInfo::randSupported

CPA_TRUE if instance supports the random bit/number generation API. See **Random Bit/Number Generation API**.

CpaBoolean _CpaCyCapabilitiesInfo::kptSupported

CPA_TRUE if instance supports the Intel(R) KPT Cryptographic API. See **Intel(R) Key Protection Technology (KPT) Cryptographic API**.

7.6 Typedef Documentation

typedef struct **_CpaCyCapabilitiesInfo CpaCyCapabilitiesInfo**

File: cpa_cy_im.h

Cryptographic Capabilities Info

This structure contains the capabilities that vary across API implementations. This structure is used in conjunction with **cpaCyQueryCapabilities()** to determine the capabilities supported by a particular API implementation.

The client MUST allocate memory for this structure and any members that require memory. When the structure is passed into the function ownership of the memory passes to the function. Ownership of the memory returns to the client when the function returns.

7.7 Function Documentation

CpaStatus cpaCyStartInstance (**CpaInstanceHandle** *instanceHandle*)

File: cpa_cy_im.h

Cryptographic Component Initialization and Start function.

This function will initialize and start the Cryptographic component. It MUST be called before any other crypto function is called. This function SHOULD be called only once (either for the very first time, or after an cpaCyStopInstance call which succeeded) per instance. Subsequent calls will have no effect.

Context:

This function may sleep, and MUST NOT be called in interrupt context.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:
`[out]` *instanceHandle* Handle to an instance of this API to be initialized.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed. Suggested course of action is to shutdown and restart.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
None.

Postcondition:
None

Note:
Note that this is a synchronous function and has no completion callback associated with it.

See also:
cpaCyStopInstance()

CpaStatus cpaCyStopInstance (**CpaInstanceHandle** *instanceHandle*)

File: cpa_cy_im.h

Cryptographic Component Stop function.

This function will stop the Cryptographic component and free all system resources associated with it. The client MUST ensure that all outstanding operations have completed before calling this function. The recommended approach to ensure this is to deregister all session or callback handles before calling this function. If outstanding operations still exist when this function is invoked, the callback function for each of those operations will NOT be invoked and the shutdown will continue. If the component is to be restarted, then a call to cpaCyStartInstance is required.

Context:
This function may sleep, and so MUST NOT be called in interrupt context.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:
`[in]` *instanceHandle* Handle to an instance of this API to be shutdown.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed. Suggested course of action is to ensure requests are not still being submitted and that all sessions are deregistered. If this does not help, then forcefully remove the component from the system.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance.

Postcondition:
None

Note:
Note that this is a synchronous function and has no completion callback associated with it.

See also:
cpaCyStartInstance()

CpaStatus cpaCyQueryCapabilities (const **CpaInstanceHandle** *instanceHandle*,
CpaCyCapabilitiesInfo * *pCapInfo*
)

File: cpa_cy_im.h

Returns capabilities of a Cryptographic API instance

This function is used to query the instance capabilities.

Context:
The function shall not be called in an interrupt context.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Handle to an instance of this API.
[out]	*pCapInfo*	Pointer to capabilities info structure. All fields in the structure are populated by the API instance.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The instance has been initialized via the **cpaCyStartInstance** function.

Postcondition:
None

CpaStatus cpaCySetAddressTranslation (const **CpaInstanceHandle** *instanceHandle*,
CpaVirtualToPhysical *virtual2Physical*
)

File: cpa_cy_im.h

Sets the address translation function

This function is used to set the virtual to physical address translation routine for the instance. The specified routine is used by the instance to perform any required translation of a virtual address to a physical address. If the application does not invoke this function, then the instance will use its default method, such as virt2phys, for address translation.

Context:
The function shall not be called in an interrupt context.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Handle to an instance of this API.
`[in]`	*virtual2Physical*	Routine that performs virtual to physical address translation.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
None

Postcondition:
None

See also:
None

8 Symmetric Cipher and Hash Cryptographic API

[Cryptographic API]

Collaboration diagram for Symmetric Cipher and Hash Cryptographic API:

8.1 Detailed Description

File: cpa_cy_sym.h

These functions specify the Cryptographic API for symmetric cipher, hash, and combined cipher and hash operations.

8.2 Modules

- **Symmetric cryptographic Data Plane API**

8.3 Data Structures

- struct **_CpaCySymCipherSetupData**
- struct **_CpaCySymHashNestedModeSetupData**
- struct **_CpaCySymHashAuthModeSetupData**
- struct **_CpaCySymHashSetupData**
- struct **_CpaCySymSessionSetupData**
- struct **_CpaCySymSessionUpdateData**
- struct **_CpaCySymOpData**
- struct **_CpaCySymStats**
- struct **_CpaCySymStats64**
- struct **_CpaCySymCapabilitiesInfo**

8.4 Defines

- #define **CPA_CY_SYM_CIPHER_CAP_BITMAP_SIZE**
- #define **CPA_CY_SYM_HASH_CAP_BITMAP_SIZE**
- #define **CPA_CY_SYM_CCM_SET_NONCE**(pOpData, pNonce, nonceLen)
- #define **CPA_CY_SYM_CCM_SET_AAD**(pOpData, pAad, aadLen)

8.5 Typedefs

- typedef void * **CpaCySymSessionCtx**
- typedef enum **_CpaCySymPacketType CpaCySymPacketType**
- typedef enum **_CpaCySymOp CpaCySymOp**
- typedef enum **_CpaCySymCipherAlgorithm CpaCySymCipherAlgorithm**
- typedef enum **_CpaCySymCipherDirection CpaCySymCipherDirection**
- typedef **_CpaCySymCipherSetupData CpaCySymCipherSetupData**
- typedef enum **_CpaCySymHashMode CpaCySymHashMode**
- typedef enum **_CpaCySymHashAlgorithm CpaCySymHashAlgorithm**
- typedef **_CpaCySymHashNestedModeSetupData CpaCySymHashNestedModeSetupData**

- typedef **_CpaCySymHashAuthModeSetupData CpaCySymHashAuthModeSetupData**
- typedef **_CpaCySymHashSetupData CpaCySymHashSetupData**
- typedef enum **_CpaCySymAlgChainOrder CpaCySymAlgChainOrder**
- typedef **_CpaCySymSessionSetupData CpaCySymSessionSetupData**
- typedef **_CpaCySymSessionUpdateData CpaCySymSessionUpdateData**
- typedef **_CpaCySymOpData CpaCySymOpData**
- typedef **_CpaCySymStats CPA_DEPRECATED**
- typedef **_CpaCySymStats64 CpaCySymStats64**
- typedef void(* **CpaCySymCbFunc**)(void *pCallbackTag, **CpaStatus** status, const **CpaCySymOp** operationType, void *pOpData, **CpaBufferList** *pDstBuffer, **CpaBoolean** verifyResult)
- typedef **_CpaCySymCapabilitiesInfo CpaCySymCapabilitiesInfo**

8.6 Enumerations

- enum **_CpaCySymPacketType** {
  CPA_CY_SYM_PACKET_TYPE_FULL,
  CPA_CY_SYM_PACKET_TYPE_PARTIAL,
  CPA_CY_SYM_PACKET_TYPE_LAST_PARTIAL
  }
- enum **_CpaCySymOp** {
  CPA_CY_SYM_OP_NONE,
  CPA_CY_SYM_OP_CIPHER,
  CPA_CY_SYM_OP_HASH,
  CPA_CY_SYM_OP_ALGORITHM_CHAINING
  }
- enum **_CpaCySymCipherAlgorithm** {
  CPA_CY_SYM_CIPHER_NULL,
  CPA_CY_SYM_CIPHER_ARC4,
  CPA_CY_SYM_CIPHER_AES_ECB,
  CPA_CY_SYM_CIPHER_AES_CBC,
  CPA_CY_SYM_CIPHER_AES_CTR,
  CPA_CY_SYM_CIPHER_AES_CCM,
  CPA_CY_SYM_CIPHER_AES_GCM,
  CPA_CY_SYM_CIPHER_DES_ECB,
  CPA_CY_SYM_CIPHER_DES_CBC,
  CPA_CY_SYM_CIPHER_3DES_ECB,
  CPA_CY_SYM_CIPHER_3DES_CBC,
  CPA_CY_SYM_CIPHER_3DES_CTR,
  CPA_CY_SYM_CIPHER_KASUMI_F8,
  CPA_CY_SYM_CIPHER_SNOW3G_UEA2,
  CPA_CY_SYM_CIPHER_AES_F8,
  CPA_CY_SYM_CIPHER_AES_XTS,
  CPA_CY_SYM_CIPHER_ZUC_EEA3
  }
- enum **_CpaCySymCipherDirection** {
  CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT,
  CPA_CY_SYM_CIPHER_DIRECTION_DECRYPT
  }
- enum **_CpaCySymHashMode** {
  CPA_CY_SYM_HASH_MODE_PLAIN,
  CPA_CY_SYM_HASH_MODE_AUTH,
  CPA_CY_SYM_HASH_MODE_NESTED
  }
- enum **_CpaCySymHashAlgorithm** {
  CPA_CY_SYM_HASH_NONE,
  CPA_CY_SYM_HASH_MD5,

CPA_CY_SYM_HASH_SHA1,
CPA_CY_SYM_HASH_SHA224,
CPA_CY_SYM_HASH_SHA256,
CPA_CY_SYM_HASH_SHA384,
CPA_CY_SYM_HASH_SHA512,
CPA_CY_SYM_HASH_AES_XCBC,
CPA_CY_SYM_HASH_AES_CCM,
CPA_CY_SYM_HASH_AES_GCM,
CPA_CY_SYM_HASH_KASUMI_F9,
CPA_CY_SYM_HASH_SNOW3G_UIA2,
CPA_CY_SYM_HASH_AES_CMAC,
CPA_CY_SYM_HASH_AES_GMAC,
CPA_CY_SYM_HASH_AES_CBC_MAC,
CPA_CY_SYM_HASH_ZUC_EIA3,
CPA_CY_SYM_HASH_SHA3_256
}

- enum **_CpaCySymAlgChainOrder** {
CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER,
CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH
}

8.7 Functions

- **CpaStatus cpaCySymSessionCtxGetSize** (const **CpaInstanceHandle** instanceHandle, const **CpaCySymSessionSetupData** *pSessionSetupData, **Cpa32U** *pSessionCtxSizeInBytes)
- **CpaStatus cpaCySymSessionCtxGetDynamicSize** (const **CpaInstanceHandle** instanceHandle, const **CpaCySymSessionSetupData** *pSessionSetupData, **Cpa32U** *pSessionCtxSizeInBytes)
- **CpaStatus cpaCySymInitSession** (const **CpaInstanceHandle** instanceHandle, const **CpaCySymCbFunc** pSymCb, const **CpaCySymSessionSetupData** *pSessionSetupData, **CpaCySymSessionCtx** sessionCtx)
- **CpaStatus cpaCySymRemoveSession** (const **CpaInstanceHandle** instanceHandle, **CpaCySymSessionCtx** pSessionCtx)
- **CpaStatus cpaCySymUpdateSession** (**CpaCySymSessionCtx** sessionCtx, const **CpaCySymSessionUpdateData** *pSessionUpdateData)
- **CpaStatus cpaCySymSessionInUse** (**CpaCySymSessionCtx** sessionCtx, **CpaBoolean** *pSessionInUse)
- **CpaStatus cpaCySymPerformOp** (const **CpaInstanceHandle** instanceHandle, void *pCallbackTag, const **CpaCySymOpData** *pOpData, const **CpaBufferList** *pSrcBuffer, **CpaBufferList** *pDstBuffer, **CpaBoolean** *pVerifyResult)
- **CpaStatus CPA_DEPRECATED cpaCySymQueryStats** (const **CpaInstanceHandle** instanceHandle, struct **_CpaCySymStats** *pSymStats)
- **CpaStatus cpaCySymQueryStats64** (const **CpaInstanceHandle** instanceHandle, **CpaCySymStats64** *pSymStats)
- **CpaStatus cpaCySymQueryCapabilities** (const **CpaInstanceHandle** instanceHandle, **CpaCySymCapabilitiesInfo** *pCapInfo)

8.8 Data Structure Documentation

8.8.1 _CpaCySymCipherSetupData Struct Reference

8.8.1.1 Detailed Description

File: cpa_cy_sym.h

8.8.1 _CpaCySymCipherSetupData Struct Reference

Symmetric Cipher Setup Data.

This structure contains data relating to Cipher (Encryption and Decryption) to set up a session.

8.8.1.2 Data Fields

- **CpaCySymCipherAlgorithm cipherAlgorithm**
- **Cpa32U cipherKeyLenInBytes**
- **Cpa8U * pCipherKey**
- **CpaCySymCipherDirection cipherDirection**

8.8.1.3 Field Documentation

CpaCySymCipherAlgorithm _CpaCySymCipherSetupData::cipherAlgorithm

Cipher algorithm and mode

Cpa32U _CpaCySymCipherSetupData::cipherKeyLenInBytes

Cipher key length in bytes. For AES it can be 128 bits (16 bytes), 192 bits (24 bytes) or 256 bits (32 bytes). For the CCM mode of operation, the only supported key length is 128 bits (16 bytes). For the CPA_CY_SYM_CIPHER_AES_F8 mode of operation, cipherKeyLenInBytes should be set to the combined length of the encryption key and the keymask. Since the keymask and the encryption key are the same size, cipherKeyLenInBytes should be set to 2 x the AES encryption key length. For the AES-XTS mode of operation:

- Two keys must be provided and cipherKeyLenInBytes refers to total length of the two keys.
- Each key can be either 128 bits (16 bytes) or 256 bits (32 bytes).
- Both keys must have the same size.

Cpa8U* _CpaCySymCipherSetupData::pCipherKey

Cipher key For the CPA_CY_SYM_CIPHER_AES_F8 mode of operation, pCipherKey will point to a concatenation of the AES encryption key followed by a keymask. As per RFC3711, the keymask should be padded with trailing bytes to match the length of the encryption key used. For AES-XTS mode of operation, two keys must be provided and pCipherKey must point to the two keys concatenated together (Key1 || Key2). cipherKeyLenInBytes will contain the total size of both keys.

CpaCySymCipherDirection _CpaCySymCipherSetupData::cipherDirection

This parameter determines if the cipher operation is an encrypt or a decrypt operation. For the RC4 algorithm and the F8/CTR modes, only encrypt operations are valid.

8.8.2 _CpaCySymHashNestedModeSetupData Struct Reference

8.8.2.1 Detailed Description

File: cpa_cy_sym.h

Hash Mode Nested Setup Data.

This structure contains data relating to a hash session in CPA_CY_SYM_HASH_MODE_NESTED mode.

8.8.2.2 Data Fields

- **Cpa8U * pInnerPrefixData**
- **Cpa32U innerPrefixLenInBytes**
- **CpaCySymHashAlgorithm outerHashAlgorithm**
- **Cpa8U * pOuterPrefixData**
- **Cpa32U outerPrefixLenInBytes**

8.8.2.3 Field Documentation

Cpa8U* _CpaCySymHashNestedModeSetupData::pInnerPrefixData

A pointer to a buffer holding the Inner Prefix data. For optimal performance the prefix data SHOULD be 8-byte aligned. This data is prepended to the data being hashed before the inner hash operation is performed.

Cpa32U _CpaCySymHashNestedModeSetupData::innerPrefixLenInBytes

The inner prefix length in bytes. The maximum size the prefix data can be is 255 bytes.

CpaCySymHashAlgorithm _CpaCySymHashNestedModeSetupData::outerHashAlgorithm

The hash algorithm used for the outer hash. Note: The inner hash algorithm is provided in the hash context.

Cpa8U* _CpaCySymHashNestedModeSetupData::pOuterPrefixData

A pointer to a buffer holding the Outer Prefix data. For optimal performance the prefix data SHOULD be 8-byte aligned. This data is prepended to the output from the inner hash operation before the outer hash operation is performed.

Cpa32U _CpaCySymHashNestedModeSetupData::outerPrefixLenInBytes

The outer prefix length in bytes. The maximum size the prefix data can be is 255 bytes.

8.8.3 _CpaCySymHashAuthModeSetupData Struct Reference

8.8.3.1 Detailed Description

File: cpa_cy_sym.h

Hash Auth Mode Setup Data.

This structure contains data relating to a hash session in CPA_CY_SYM_HASH_MODE_AUTH mode.

8.8.3.2 Data Fields

- **Cpa8U * authKey**
- **Cpa32U authKeyLenInBytes**
- **Cpa32U aadLenInBytes**

8.8.3.3 Field Documentation

Cpa8U* _CpaCySymHashAuthModeSetupData::authKey

Authentication key pointer. For the GCM (**CPA_CY_SYM_HASH_AES_GCM**) and CCM (**CPA_CY_SYM_HASH_AES_CCM**) modes of operation, this field is ignored; the authentication key is the same as the cipher key (see the field pCipherKey in struct **CpaCySymCipherSetupData**).

Cpa32U _CpaCySymHashAuthModeSetupData::authKeyLenInBytes

Length of the authentication key in bytes. The key length MUST be less than or equal to the block size of the algorithm. It is the client's responsibility to ensure that the key length is compliant with the standard being used (for example RFC 2104, FIPS 198a).

For the GCM (**CPA_CY_SYM_HASH_AES_GCM**) and CCM (**CPA_CY_SYM_HASH_AES_CCM**) modes of operation, this field is ignored; the authentication key is the same as the cipher key, and so is its length (see the field cipherKeyLenInBytes in struct **CpaCySymCipherSetupData**).

Cpa32U _CpaCySymHashAuthModeSetupData::aadLenInBytes

The length of the additional authenticated data (AAD) in bytes. The maximum permitted value is 240 bytes, unless otherwise specified below.

This field must be specified when the hash algorithm is one of the following:

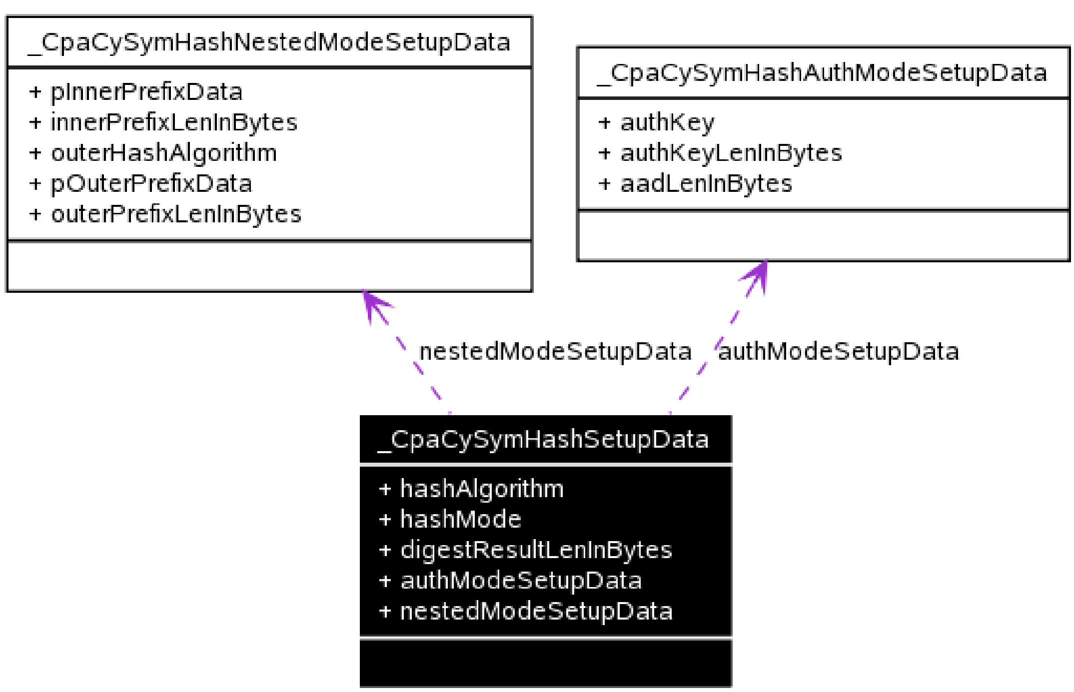
- For SNOW3G (**CPA_CY_SYM_HASH_SNOW3G_UIA2**), this is the length of the IV (which should be 16).
- For GCM (**CPA_CY_SYM_HASH_AES_GCM**). In this case, this is the length of the Additional Authenticated Data (called A, in NIST SP800-38D).
- For CCM (**CPA_CY_SYM_HASH_AES_CCM**). In this case, this is the length of the associated data (called A, in NIST SP800-38C). Note that this does NOT include the length of any padding, or the 18 bytes reserved at the start of the above field to store the block B0 and the encoded length. The maximum permitted value in this case is 222 bytes.

Note:

For AES-GMAC (**CPA_CY_SYM_HASH_AES_GMAC**) mode of operation this field is not used and should be set to 0. Instead the length of the AAD data is specified in the messageLenToHashInBytes field of the CpaCySymOpData structure.

8.8.4 _CpaCySymHashSetupData Struct Reference

Collaboration diagram for _CpaCySymHashSetupData:



8.8.4 _CpaCySymHashSetupData Struct Reference

8.8.4.1 Detailed Description

File: cpa_cy_sym.h

Hash Setup Data.

This structure contains data relating to a hash session. The fields hashAlgorithm, hashMode and digestResultLenInBytes are common to all three hash modes and MUST be set for each mode.

8.8.4.2 Data Fields

- **CpaCySymHashAlgorithm hashAlgorithm**
- **CpaCySymHashMode hashMode**
- **Cpa32U digestResultLenInBytes**
- **CpaCySymHashAuthModeSetupData authModeSetupData**
- **CpaCySymHashNestedModeSetupData nestedModeSetupData**

8.8.4.3 Field Documentation

CpaCySymHashAlgorithm _CpaCySymHashSetupData::hashAlgorithm

Hash algorithm. For mode CPA_CY_SYM_MODE_HASH_NESTED, this is the inner hash algorithm.

CpaCySymHashMode _CpaCySymHashSetupData::hashMode

Mode of the hash operation. Valid options include plain, auth or nested hash mode.

Cpa32U _CpaCySymHashSetupData::digestResultLenInBytes

Length of the digest to be returned. If the verify option is set, this specifies the length of the digest to be compared for the session.

For CCM (**CPA_CY_SYM_HASH_AES_CCM**), this is the octet length of the MAC, which can be one of 4, 6, 8, 10, 12, 14 or 16.

For GCM (**CPA_CY_SYM_HASH_AES_GCM**), this is the length in bytes of the authentication tag.

If the value is less than the maximum length allowed by the hash, the result shall be truncated. If the value is greater than the maximum length allowed by the hash, an error (**CPA_STATUS_INVALID_PARAM**) is returned from the function **cpaCySymInitSession**.

In the case of nested hash, it is the outer hash which determines the maximum length allowed.

CpaCySymHashAuthModeSetupData _CpaCySymHashSetupData::authModeSetupData

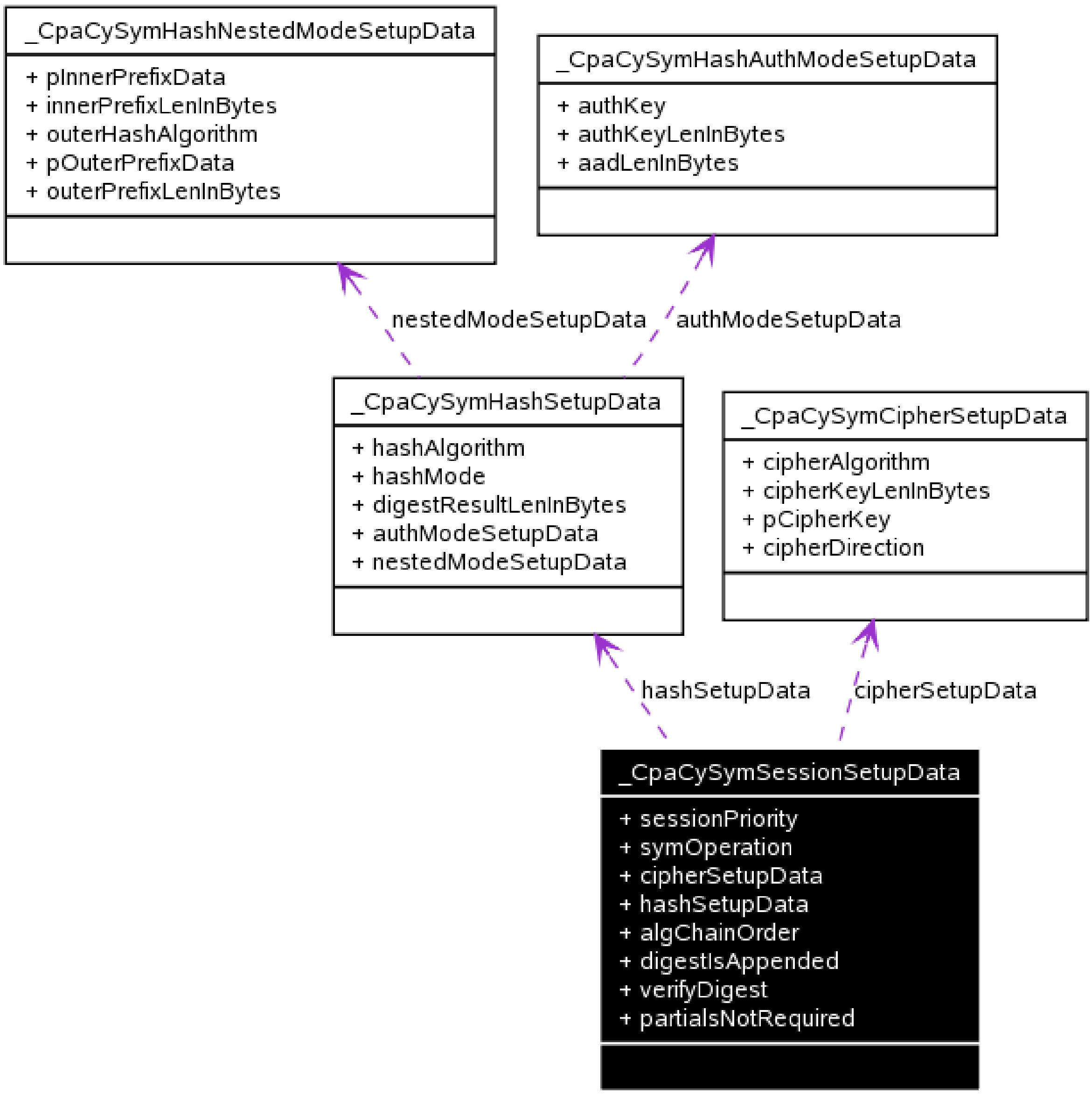
Authentication Mode Setup Data. Only valid for mode CPA_CY_SYM_MODE_HASH_AUTH

CpaCySymHashNestedModeSetupData _CpaCySymHashSetupData::nestedModeSetupData

Nested Hash Mode Setup Data Only valid for mode CPA_CY_SYM_MODE_HASH_NESTED

8.8.5 _CpaCySymSessionSetupData Struct Reference

Collaboration diagram for _CpaCySymSessionSetupData:



8.8.5.1 Detailed Description

File: cpa_cy_sym.h

Session Setup Data.

This structure contains data relating to setting up a session. The client needs to complete the information in this structure in order to setup a session.

8.8.5.2 Data Fields

- **CpaCyPriority sessionPriority**
- **CpaCySymOp symOperation**

- **CpaCySymCipherSetupData cipherSetupData**
- **CpaCySymHashSetupData hashSetupData**
- **CpaCySymAlgChainOrder algChainOrder**
- **CpaBoolean digestIsAppended**
- **CpaBoolean verifyDigest**
- **CpaBoolean partialsNotRequired**

8.8.5.3 Field Documentation

CpaCyPriority _CpaCySymSessionSetupData::sessionPriority

Priority of this session

CpaCySymOp _CpaCySymSessionSetupData::symOperation

Operation to perfom

CpaCySymCipherSetupData _CpaCySymSessionSetupData::cipherSetupData

Cipher Setup Data for the session. This member is ignored for the CPA_CY_SYM_OP_HASH operation.

CpaCySymHashSetupData _CpaCySymSessionSetupData::hashSetupData

Hash Setup Data for a session. This member is ignored for the CPA_CY_SYM_OP_CIPHER operation.

CpaCySymAlgChainOrder _CpaCySymSessionSetupData::algChainOrder

If this operation data structure relates to an algorithm chaining session then this parameter determines the order in which the chained operations are performed. If this structure does not relate to an algorithm chaining session then this parameter will be ignored.

Note:

In the case of authenticated ciphers (GCM and CCM), which are also presented as "algorithm chaining", this value is also ignored. The chaining order is defined by the authenticated cipher, in those cases.

CpaBoolean _CpaCySymSessionSetupData::digestIsAppended

Flag indicating whether the digest is appended immediately following the region over which the digest is computed. This is true for both IPsec packets and SSL/TLS records.

If this flag is set, then the value of the pDigestResult field of the structure **CpaCySymOpData** is ignored.

Note:

The value of this field is ignored for the authenticated cipher AES_CCM as the digest must be appended in this case.

Setting digestIsAppended for hash only operations when verifyDigest is also set is not supported. For hash only operations when verifyDigest is set, digestIsAppended should be set to CPA_FALSE.

CpaBoolean _CpaCySymSessionSetupData::verifyDigest

This flag is relevant only for operations which generate a message digest. If set to true, the computed digest will not be written back to the buffer location specified by other parameters, but instead will be verified (i.e. compared to the value passed in at that location). The number of bytes to be written or compared is indicated by the digest output length for the session.

Note:

This option is only valid for full packets and for final partial packets when using partials without algorithm chaining.

The value of this field is ignored for the authenticated ciphers (AES_CCM and AES_GCM). Digest verification is always done for these (when the direction is decrypt) and unless the DP API is used, the message buffer will be zeroed if verification fails. When using the DP API, it is the API clients responsibility to clear the message buffer when digest verification fails.

CpaBoolean _CpaCySymSessionSetupData::partialsNotRequired

This flag indicates if partial packet processing is required for this session. If set to true, partial packet processing will not be enabled for this session and any calls to **cpaCySymPerformOp()** with the packetType parameter set to a value other than CPA_CY_SYM_PACKET_TYPE_FULL will fail.

8.8.6 _CpaCySymSessionUpdateData Struct Reference

8.8.6.1 Detailed Description

File: cpa_cy_sym.h

Session Update Data.

This structure contains data relating to resetting a session.

8.8.6.2 Data Fields

- **Cpa32U flags**
- **Cpa8U * pCipherKey**
- **CpaCySymCipherDirection cipherDirection**
- **Cpa8U * authKey**

8.8.6.3 Field Documentation

Cpa32U _CpaCySymSessionUpdateData::flags

Flags indicating which fields to update. All bits should be set to 0 except those fields to be updated.

Cpa8U* _CpaCySymSessionUpdateData::pCipherKey

Cipher key. The same restrictions apply as described in the corresponding field of the data structure **CpaCySymCipherSetupData**.

CpaCySymCipherDirection _CpaCySymSessionUpdateData::cipherDirection

This parameter determines if the cipher operation is an encrypt or a decrypt operation. The same restrictions apply as described in the corresponding field of the data structure **CpaCySymCipherSetupData**.

Cpa8U* _CpaCySymSessionUpdateData::authKey

Authentication key pointer. The same restrictions apply as described in the corresponding field of the data structure **CpaCySymHashAuthModeSetupData**.

8.8.7 _CpaCySymOpData Struct Reference

8.8.7.1 Detailed Description

File: cpa_cy_sym.h

8.8.7 _CpaCySymOpData Struct Reference

Cryptographic Component Operation Data.

This structure contains data relating to performing cryptographic processing on a data buffer. This request is used with **cpaCySymPerformOp()** call for performing cipher, hash, auth cipher or a combined hash and cipher operation.

See also:
  CpaCySymPacketType

Note:
  If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCySymPerformOp function, and before it has been returned in the callback, undefined behavior will result.

8.8.7.2 Data Fields

- **CpaCySymSessionCtx sessionCtx**
- **CpaCySymPacketType packetType**
- **Cpa8U * pIv**
- **Cpa32U ivLenInBytes**
- **Cpa32U cryptoStartSrcOffsetInBytes**
- **Cpa32U messageLenToCipherInBytes**
- **Cpa32U hashStartSrcOffsetInBytes**
- **Cpa32U messageLenToHashInBytes**
- **Cpa8U * pDigestResult**
- **Cpa8U * pAdditionalAuthData**

8.8.7.3 Field Documentation

CpaCySymSessionCtx _CpaCySymOpData::sessionCtx

Handle for the initialized session context

CpaCySymPacketType _CpaCySymOpData::packetType

Selects the packet type

Cpa8U* _CpaCySymOpData::pIv

Initialization Vector or Counter.

- For block ciphers in CBC or F8 mode, or for Kasumi in F8 mode, or for SNOW3G in UEA2 mode, this is the Initialization Vector (IV) value.
- For block ciphers in CTR mode, this is the counter.
- For GCM mode, this is either the IV (if the length is 96 bits) or J0 (for other sizes), where J0 is as defined by NIST SP800-38D. Regardless of the IV length, a full 16 bytes needs to be allocated.
- For CCM mode, the first byte is reserved, and the nonce should be written starting at &pIv[1] (to allow space for the implementation to write in the flags in the first byte). Note that a full 16 bytes should be allocated, even though the ivLenInBytes field will have a value less than this. The macro **CPA_CY_SYM_CCM_SET_NONCE** may be used here.
- For AES-XTS, this is the 128bit tweak, i, from IEEE Std 1619-2007.

For optimum performance, the data pointed to SHOULD be 8-byte aligned.

The IV/Counter will be updated after every partial cryptographic operation.

Cpa32U _CpaCySymOpData::ivLenInBytes

Length of valid IV data pointed to by the pIv parameter.

- For block ciphers in CBC or F8 mode, or for Kasumi in F8 mode, or for SNOW3G in UEA2 mode, this is the length of the IV (which must be the same as the block length of the cipher).
- For block ciphers in CTR mode, this is the length of the counter (which must be the same as the block length of the cipher).
- For GCM mode, this is either 12 (for 96-bit IVs) or 16, in which case pIv points to J0.
- For CCM mode, this is the length of the nonce, which can be in the range 7 to 13 inclusive.

Cpa32U _CpaCySymOpData::cryptoStartSrcOffsetInBytes

Starting point for cipher processing, specified as number of bytes from start of data in the source buffer. The result of the cipher operation will be written back into the output buffer starting at this location.

Cpa32U _CpaCySymOpData::messageLenToCipherInBytes

The message length, in bytes, of the source buffer on which the cryptographic operation will be computed. This must be a multiple of the block size if a block cipher is being used. This is also the same as the result length.

Note:

In the case of CCM (**CPA_CY_SYM_HASH_AES_CCM**), this value should not include the length of the padding or the length of the MAC; the driver will compute the actual number of bytes over which the encryption will occur, which will include these values.

There are limitations on this length for partial operations. Refer to the cpaCySymPerformOp function description for details.

On some implementations, this length may be limited to a 16-bit value (65535 bytes).

For AES-GMAC (**CPA_CY_SYM_HASH_AES_GMAC**), this field should be set to 0.

Cpa32U _CpaCySymOpData::hashStartSrcOffsetInBytes

Starting point for hash processing, specified as number of bytes from start of packet in source buffer.

Note:

For CCM and GCM modes of operation, this field is ignored. The field **pAdditionalAuthData** field should be set instead.

For AES-GMAC (**CPA_CY_SYM_HASH_AES_GMAC**) mode of operation, this field specifies the start of the AAD data in the source buffer.

Cpa32U _CpaCySymOpData::messageLenToHashInBytes

The message length, in bytes, of the source buffer that the hash will be computed on.

Note:

There are limitations on this length for partial operations. Refer to the **cpaCySymPerformOp** function description for details.

For CCM and GCM modes of operation, this field is ignored. The field **pAdditionalAuthData** field should be set instead.

For AES-GMAC (**CPA_CY_SYM_HASH_AES_GMAC**) mode of operation, this field specifies the length of the AAD data in the source buffer.

On some implementations, this length may be limited to a 16-bit value (65535 bytes).

Cpa8U* _CpaCySymOpData::pDigestResult

If the digestIsAppended member of the **CpaCySymSessionSetupData** structure is NOT set then this is a pointer to the location where the digest result should be inserted (in the case of digest generation) or where the purported digest exists (in the case of digest verification).

At session registration time, the client specified the digest result length with the digestResultLenInBytes member of the **CpaCySymHashSetupData** structure. The client must allocate at least digestResultLenInBytes of physically contiguous memory at this location.

For partial packet processing without algorithm chaining, this pointer will be ignored for all but the final partial operation.

For digest generation, the digest result will overwrite any data at this location.

Note:

For GCM (**CPA_CY_SYM_HASH_AES_GCM**), for "digest result" read "authentication tag T".

If the digestIsAppended member of the **CpaCySymSessionSetupData** structure is set then this value is ignored and the digest result is understood to be in the destination buffer for digest generation, and in the source buffer for digest verification. The location of the digest result in this case is immediately following the region over which the digest is computed.

Cpa8U* _CpaCySymOpData::pAdditionalAuthData

Pointer to Additional Authenticated Data (AAD) needed for authenticated cipher mechanisms (CCM and GCM), and to the IV for SNOW3G authentication (**CPA_CY_SYM_HASH_SNOW3G_UIA2**). For other authentication mechanisms this pointer is ignored.

The length of the data pointed to by this field is set up for the session in the **CpaCySymHashAuthModeSetupData** structure as part of the **cpaCySymInitSession** function call. This length must not exceed 240 bytes.

Specifically for CCM (**CPA_CY_SYM_HASH_AES_CCM**), the caller should setup this field as follows:

- the nonce should be written starting at an offset of one byte into the array, leaving room for the implementation to write in the flags to the first byte. For example,
  memcpy(&pOpData->pAdditionalAuthData[1], pNonce, nonceLen);
  The macro **CPA_CY_SYM_CCM_SET_NONCE** may be used here.

- the additional authentication data itself should be written starting at an offset of 18 bytes into the array, leaving room for the length encoding in the first two bytes of the second block. For example,
  memcpy(&pOpData->pAdditionalAuthData[18], pAad, aadLen);
  The macro **CPA_CY_SYM_CCM_SET_AAD** may be used here.

- the array should be big enough to hold the above fields, plus any padding to round this up to the nearest multiple of the block size (16 bytes). Padding will be added by the implementation.

Finally, for GCM (**CPA_CY_SYM_HASH_AES_GCM**), the caller should setup this field as follows:

- the AAD is written in starting at byte 0
- the array must be big enough to hold the AAD, plus any padding to round this up to the nearest multiple of the block size (16 bytes). Padding will be added by the implementation.

Note:

For AES-GMAC (**CPA_CY_SYM_HASH_AES_GMAC**) mode of operation, this field is not used and should be set to 0. Instead the AAD data should be placed in the source buffer.

8.8.8 _CpaCySymStats Struct Reference

8.8.8.1 Detailed Description

File: cpa_cy_sym.h

Cryptographic Component Statistics.

Deprecated:
As of v1.3 of the cryptographic API, this structure has been deprecated, replaced by **CpaCySymStats64**.

This structure contains statistics on the Symmetric Cryptographic operations. Statistics are set to zero when the component is initialized.

8.8.8.2 Data Fields

- **Cpa32U numSessionsInitialized**
- **Cpa32U numSessionsRemoved**
- **Cpa32U numSessionErrors**
- **Cpa32U numSymOpRequests**
- **Cpa32U numSymOpRequestErrors**
- **Cpa32U numSymOpCompleted**
- **Cpa32U numSymOpCompletedErrors**
- **Cpa32U numSymOpVerifyFailures**

8.8.8.3 Field Documentation

Cpa32U _CpaCySymStats::numSessionsInitialized

Number of session initialized

Cpa32U _CpaCySymStats::numSessionsRemoved

Number of sessions removed

Cpa32U _CpaCySymStats::numSessionErrors

Number of session initialized and removed errors.

Cpa32U _CpaCySymStats::numSymOpRequests

Number of successful symmetric operation requests.

Cpa32U _CpaCySymStats::numSymOpRequestErrors

Number of operation requests that had an error and could not be processed.

Cpa32U _CpaCySymStats::numSymOpCompleted

Number of operations that completed successfully.

Cpa32U _CpaCySymStats::numSymOpCompletedErrors

Number of operations that could not be completed successfully due to errors.

Cpa32U _CpaCySymStats::numSymOpVerifyFailures

Number of operations that completed successfully, but the result of the digest verification test was that it failed. Note that this does not indicate an error condition.

8.8.9 _CpaCySymStats64 Struct Reference

8.8.9.1 Detailed Description

File: cpa_cy_sym.h

Cryptographic Component Statistics (64-bit version).

This structure contains a 64-bit version of the statistics on the Symmetric Cryptographic operations. Statistics are set to zero when the component is initialized.

8.8.9.2 Data Fields

- **Cpa64U numSessionsInitialized**
- **Cpa64U numSessionsRemoved**
- **Cpa64U numSessionErrors**
- **Cpa64U numSymOpRequests**
- **Cpa64U numSymOpRequestErrors**
- **Cpa64U numSymOpCompleted**
- **Cpa64U numSymOpCompletedErrors**
- **Cpa64U numSymOpVerifyFailures**

8.8.9.3 Field Documentation

Cpa64U _CpaCySymStats64::numSessionsInitialized

Number of session initialized

Cpa64U _CpaCySymStats64::numSessionsRemoved

Number of sessions removed

Cpa64U _CpaCySymStats64::numSessionErrors

Number of session initialized and removed errors.

Cpa64U _CpaCySymStats64::numSymOpRequests

Number of successful symmetric operation requests.

Cpa64U _CpaCySymStats64::numSymOpRequestErrors

Number of operation requests that had an error and could not be processed.

Cpa64U _CpaCySymStats64::numSymOpCompleted

Number of operations that completed successfully.

Cpa64U _CpaCySymStats64::numSymOpCompletedErrors

Number of operations that could not be completed successfully due to errors.

Cpa64U _CpaCySymStats64::numSymOpVerifyFailures

Number of operations that completed successfully, but the result of the digest verification test was that it failed. Note that this does not indicate an error condition.

8.8.10 _CpaCySymCapabilitiesInfo Struct Reference

8.8.10.1 Detailed Description

File: cpa_cy_sym.h

Symmetric Capabilities Info

This structure contains the capabilities that vary across implementations of the symmetric sub-API of the cryptographic API. This structure is used in conjunction with **cpaCySymQueryCapabilities()** to determine the capabilities supported by a particular API implementation.

For example, to see if an implementation supports cipher **CPA_CY_SYM_CIPHER_AES_CBC**, use the code

```
if (CPA_BITMAP_BIT_TEST(capInfo.ciphers, CPA_CY_SYM_CIPHER_AES_CBC))
{
    // algo is supported
}
else
{
    // algo is not supported
}
```

The client MUST allocate memory for this structure and any members that require memory. When the structure is passed into the function ownership of the memory passes to the function. Ownership of the memory returns to the client when the function returns.

8.8.10.2 Public Member Functions

- **CPA_BITMAP** (ciphers, CPA_CY_SYM_CIPHER_CAP_BITMAP_SIZE)
- **CPA_BITMAP** (hashes, CPA_CY_SYM_HASH_CAP_BITMAP_SIZE)

8.8.10.3 Data Fields

- **CpaBoolean partialPacketSupported**

8.8.10.4 Member Function Documentation

_CpaCySymCapabilitiesInfo::CPA_BITMAP (	ciphers CPA_CY_SYM_CIPHER_CAP_BITMAP_SIZE	,
)		

Bitmap representing which cipher algorithms (and modes) are supported by the instance. Bits can be tested using the macro **CPA_BITMAP_BIT_TEST**. The bit positions are those specified in the enumerated type **CpaCySymCipherAlgorithm**.

_CpaCySymCapabilitiesInfo::CPA_BITMAP (	hashes CPA_CY_SYM_HASH_CAP_BITMAP_SIZE	,
)		

Bitmap representing which hash/authentication algorithms are supported by the instance. Bits can be tested using the macro **CPA_BITMAP_BIT_TEST**. The bit positions are those specified in the enumerated type **CpaCySymHashAlgorithm**.

8.8.10.5 Field Documentation

CpaBoolean _CpaCySymCapabilitiesInfo::partialPacketSupported

CPA_TRUE if instance supports partial packets. See **CpaCySymPacketType**.

8.9 Define Documentation

#define CPA_CY_SYM_CIPHER_CAP_BITMAP_SIZE

File: cpa_cy_sym.h

Size of bitmap needed for cipher "capabilities" type.

Defines the number of bits in the bitmap to represent supported ciphers in the type **CpaCySymCapabilitiesInfo**. Should be set to at least one greater than the largest value in the enumerated type **CpaCySymHashAlgorithm**, so that the value of the enum constant can also be used as the bit position in the bitmap.

A larger value was chosen to allow for extensibility without the need to change the size of the bitmap (to ease backwards compatibility in future versions of the API).

#define CPA_CY_SYM_HASH_CAP_BITMAP_SIZE

File: cpa_cy_sym.h

Size of bitmap needed for hash "capabilities" type.

Defines the number of bits in the bitmap to represent supported hashes in the type **CpaCySymCapabilitiesInfo**. Should be set to at least one greater than the largest value in the enumerated type **CpaCySymHashAlgorithm**, so that the value of the enum constant can also be used as the bit position in the bitmap.

A larger value was chosen to allow for extensibility without the need to change the size of the bitmap (to ease backwards compatibility in future versions of the API).

```
#define CPA_CY_SYM_CCM_SET_NONCE ( pOpData,
                                   pNonce,
                                   nonceLen  )
```

File: cpa_cy_sym.h

Setup the nonce for CCM.

This macro sets the nonce in the appropriate locations of the **CpaCySymOpData** struct for the authenticated encryption algorithm **CPA_CY_SYM_HASH_AES_CCM**.

```
#define CPA_CY_SYM_CCM_SET_AAD ( pOpData,
                                 pAad,
                                 aadLen    )
```

File: cpa_cy_sym.h

Setup the additional authentication data for CCM.

This macro sets the additional authentication data in the appropriate location of the**CpaCySymOpData** struct for the authenticated encryptionalgorithm **CPA_CY_SYM_HASH_AES_CCM**.

8.10 Typedef Documentation

typedef void* **CpaCySymSessionCtx**

File: cpa_cy_sym.h

Cryptographic component symmetric session context handle.

Handle to a cryptographic session context. The memory for this handle is allocated by the client. The size of the memory that the client needs to allocate is determined by a call to the **cpaCySymSessionCtxGetSize** or **cpaCySymSessionCtxGetDynamicSize** functions. The session context memory is initialized with a call to the **cpaCySymInitSession** function. This memory MUST not be freed until a call to **cpaCySymRemoveSession** has completed successfully.

typedef enum **_CpaCySymPacketType CpaCySymPacketType**

File: cpa_cy_sym.h

Packet type for the cpaCySymPerformOp function

Enumeration which is used to indicate to the symmetric cryptographic perform function on which type of packet the operation is required to be invoked. Multi-part cipher and hash operations are useful when processing needs to be performed on a message which is available to the client in multiple parts (for example due to network fragmentation of the packet).

Note:
There are some restrictions regarding the operations on which partial packet processing is supported. For details, see the function **cpaCySymPerformOp**.

See also:
cpaCySymPerformOp()

typedef enum **_CpaCySymOp CpaCySymOp**

File: cpa_cy_sym.h

Types of operations supported by the cpaCySymPerformOp function.

This enumeration lists different types of operations supported by the cpaCySymPerformOp function. The operation type is defined during session registration and cannot be changed for a session once it has been setup.

See also:
cpaCySymPerformOp

typedef enum **_CpaCySymCipherAlgorithm CpaCySymCipherAlgorithm**

File: cpa_cy_sym.h

Cipher algorithms.

This enumeration lists supported cipher algorithms and modes.

typedef enum **_CpaCySymCipherDirection CpaCySymCipherDirection**

File: cpa_cy_sym.h

Symmetric Cipher Direction

This enum indicates the cipher direction (encryption or decryption).

typedef struct **_CpaCySymCipherSetupData CpaCySymCipherSetupData**

File: cpa_cy_sym.h

Symmetric Cipher Setup Data.

This structure contains data relating to Cipher (Encryption and Decryption) to set up a session.

typedef enum **_CpaCySymHashMode CpaCySymHashMode**

File: cpa_cy_sym.h

Symmetric Hash mode

This enum indicates the Hash Mode.

typedef enum **_CpaCySymHashAlgorithm CpaCySymHashAlgorithm**

File: cpa_cy_sym.h

Hash algorithms.

This enumeration lists supported hash algorithms.

typedef struct **_CpaCySymHashNestedModeSetupData CpaCySymHashNestedModeSetupData**

File: cpa_cy_sym.h

Hash Mode Nested Setup Data.

This structure contains data relating to a hash session in CPA_CY_SYM_HASH_MODE_NESTED mode.

typedef struct **_CpaCySymHashAuthModeSetupData CpaCySymHashAuthModeSetupData**

File: cpa_cy_sym.h

Hash Auth Mode Setup Data.

This structure contains data relating to a hash session in CPA_CY_SYM_HASH_MODE_AUTH mode.

typedef struct **_CpaCySymHashSetupData CpaCySymHashSetupData**

File: cpa_cy_sym.h

Hash Setup Data.

This structure contains data relating to a hash session. The fields hashAlgorithm, hashMode and digestResultLenInBytes are common to all three hash modes and MUST be set for each mode.

typedef enum **_CpaCySymAlgChainOrder CpaCySymAlgChainOrder**

File: cpa_cy_sym.h

Algorithm Chaining Operation Ordering

This enum defines the ordering of operations for algorithm chaining.

typedef struct **_CpaCySymSessionSetupData CpaCySymSessionSetupData**

File: cpa_cy_sym.h

Session Setup Data.

This structure contains data relating to setting up a session. The client needs to complete the information in this structure in order to setup a session.

typedef struct **_CpaCySymSessionUpdateData CpaCySymSessionUpdateData**

File: cpa_cy_sym.h

Session Update Data.

This structure contains data relating to resetting a session.

typedef struct **_CpaCySymOpData CpaCySymOpData**

File: cpa_cy_sym.h

Cryptographic Component Operation Data.

This structure contains data relating to performing cryptographic processing on a data buffer. This request is used with **cpaCySymPerformOp()** call for performing cipher, hash, auth cipher or a combined hash and cipher operation.

See also:
CpaCySymPacketType

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCySymPerformOp function, and before it has been returned in the callback, undefined behavior will result.

typedef struct **_CpaCySymStats CPA_DEPRECATED**

File: cpa_cy_sym.h

Cryptographic Component Statistics.

Deprecated:
As of v1.3 of the cryptographic API, this structure has been deprecated, replaced by **CpaCySymStats64**.

This structure contains statistics on the Symmetric Cryptographic operations. Statistics are set to zero when the component is initialized.

typedef struct **_CpaCySymStats64 CpaCySymStats64**

File: cpa_cy_sym.h

Cryptographic Component Statistics (64-bit version).

This structure contains a 64-bit version of the statistics on the Symmetric Cryptographic operations. Statistics are set to zero when the component is initialized.

typedef void(* **CpaCySymCbFunc**)(void *pCallbackTag, **CpaStatus** status, const **CpaCySymOp** operationType, void *pOpData, **CpaBufferList** *pDstBuffer, **CpaBoolean** verifyResult)

File: cpa_cy_sym.h

Definition of callback function

This is the callback function prototype. The callback function is registered by the application using the **cpaCySymInitSession()** function call.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*pCallbackTag*	Opaque value provided by user while making individual function call.
`[in]`	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
`[in]`	*operationType*	Identifies the operation type that was requested in the cpaCySymPerformOp function.
`[in]`	*pOpData*	Pointer to structure with input parameters.
`[in]`	*pDstBuffer*	Caller MUST allocate a sufficiently sized destination buffer to hold the data output. For out-of-place processing the data outside the cryptographic regions in the source buffer are copied into the destination buffer. To perform "in-place" processing set the pDstBuffer parameter in cpaCySymPerformOp function to point at the same location as pSrcBuffer. For optimum

		performance, the data pointed to SHOULD be 8-byte aligned.
[in]	*verifyResult*	This parameter is valid when the verifyDigest option is set in the CpaCySymSessionSetupData structure. A value of CPA_TRUE indicates that the compare succeeded. A value of CPA_FALSE indicates that the compare failed for an unspecified reason.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
cpaCySymInitSession(), **cpaCySymRemoveSession()**

typedef struct **_CpaCySymCapabilitiesInfo CpaCySymCapabilitiesInfo**

File: cpa_cy_sym.h

Symmetric Capabilities Info

This structure contains the capabilities that vary across implementations of the symmetric sub-API of the cryptographic API. This structure is used in conjunction with **cpaCySymQueryCapabilities()** to determine the capabilities supported by a particular API implementation.

For example, to see if an implementation supports cipher **CPA_CY_SYM_CIPHER_AES_CBC**, use the code

```
if (CPA_BITMAP_BIT_TEST(capInfo.ciphers, CPA_CY_SYM_CIPHER_AES_CBC))
{
    // algo is supported
}
else
{
    // algo is not supported
}
```

The client MUST allocate memory for this structure and any members that require memory. When the structure is passed into the function ownership of the memory passes to the function. Ownership of the memory returns to the client when the function returns.

8.11 Enumeration Type Documentation

enum **_CpaCySymPacketType**

File: cpa_cy_sym.h

Packet type for the cpaCySymPerformOp function

Enumeration which is used to indicate to the symmetric cryptographic perform function on which type of packet the operation is required to be invoked. Multi-part cipher and hash operations are useful when processing needs to be performed on a message which is available to the client in multiple parts (for example due to network fragmentation of the packet).

Note:

There are some restrictions regarding the operations on which partial packet processing is supported. For details, see the function **cpaCySymPerformOp**.

See also:

cpaCySymPerformOp()

Enumerator:

CPA_CY_SYM_PACKET_TYPE_FULL	Perform an operation on a full packet
CPA_CY_SYM_PACKET_TYPE_PARTIAL	Perform a partial operation and maintain the state of the partial operation within the session. This is used for either the first or subsequent packets within a partial packet flow.
CPA_CY_SYM_PACKET_TYPE_LAST_PARTIAL	Complete the last part of a multi-part operation

enum **_CpaCySymOp**

File: cpa_cy_sym.h

Types of operations supported by the cpaCySymPerformOp function.

This enumeration lists different types of operations supported by the cpaCySymPerformOp function. The operation type is defined during session registration and cannot be changed for a session once it has been setup.

See also:

cpaCySymPerformOp

Enumerator:

CPA_CY_SYM_OP_NONE	No operation
CPA_CY_SYM_OP_CIPHER	Cipher only operation on the data
CPA_CY_SYM_OP_HASH	Hash only operation on the data
CPA_CY_SYM_OP_ALGORITHM_CHAINING	Chain any cipher with any hash operation. The order depends on the value in the CpaCySymAlgChainOrder enum. This value is also used for authenticated ciphers (GCM and CCM), in which case the cipherAlgorithm should take one of the values **CPA_CY_SYM_CIPHER_AES_CCM** or **CPA_CY_SYM_CIPHER_AES_GCM**, while the hashAlgorithm should take the corresponding value **CPA_CY_SYM_HASH_AES_CCM** or **CPA_CY_SYM_HASH_AES_GCM**.

enum **_CpaCySymCipherAlgorithm**

File: cpa_cy_sym.h

Cipher algorithms.

This enumeration lists supported cipher algorithms and modes.

Enumerator:

CPA_CY_SYM_CIPHER_NULL	NULL cipher algorithm. No mode applies to the NULL algorithm.
CPA_CY_SYM_CIPHER_ARC4	(A)RC4 cipher algorithm
CPA_CY_SYM_CIPHER_AES_ECB	AES algorithm in ECB mode
CPA_CY_SYM_CIPHER_AES_CBC	AES algorithm in CBC mode
CPA_CY_SYM_CIPHER_AES_CTR	AES algorithm in Counter mode
CPA_CY_SYM_CIPHER_AES_CCM	AES algorithm in CCM mode. This authenticated cipher is only supported when the hash mode is also set to CPA_CY_SYM_HASH_MODE_AUTH. When this cipher algorithm is used the CPA_CY_SYM_HASH_AES_CCM element of the CpaCySymHashAlgorithm enum MUST be used to set up the related CpaCySymHashSetupData structure in the session context.
CPA_CY_SYM_CIPHER_AES_GCM	AES algorithm in GCM mode. This authenticated cipher is only supported when the hash mode is also set to CPA_CY_SYM_HASH_MODE_AUTH. When this cipher algorithm is used the CPA_CY_SYM_HASH_AES_GCM element of the CpaCySymHashAlgorithm enum MUST be used to set up the related CpaCySymHashSetupData structure in the session context.
CPA_CY_SYM_CIPHER_DES_ECB	DES algorithm in ECB mode
CPA_CY_SYM_CIPHER_DES_CBC	DES algorithm in CBC mode
CPA_CY_SYM_CIPHER_3DES_ECB	Triple DES algorithm in ECB mode
CPA_CY_SYM_CIPHER_3DES_CBC	Triple DES algorithm in CBC mode
CPA_CY_SYM_CIPHER_3DES_CTR	Triple DES algorithm in CTR mode
CPA_CY_SYM_CIPHER_KASUMI_F8	Kasumi algorithm in F8 mode
CPA_CY_SYM_CIPHER_SNOW3G_UEA2	SNOW3G algorithm in UEA2 mode
CPA_CY_SYM_CIPHER_AES_F8	AES algorithm in F8 mode
CPA_CY_SYM_CIPHER_AES_XTS	AES algorithm in XTS mode
CPA_CY_SYM_CIPHER_ZUC_EEA3	ZUC algorithm in EEA3 mode

enum **_CpaCySymCipherDirection**

File: cpa_cy_sym.h

Symmetric Cipher Direction

This enum indicates the cipher direction (encryption or decryption).

Enumerator:

CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT	Encrypt Data
CPA_CY_SYM_CIPHER_DIRECTION_DECRYPT	Decrypt Data

enum **_CpaCySymHashMode**

File: cpa_cy_sym.h

Symmetric Hash mode

This enum indicates the Hash Mode.

Enumerator:

CPA_CY_SYM_HASH_MODE_PLAIN	Plain hash. Can be specified for MD5 and the SHA family of hash algorithms.
CPA_CY_SYM_HASH_MODE_AUTH	Authenticated hash. This mode may be used in conjunction with the MD5 and SHA family of algorithms to specify HMAC. It MUST also be specified with all of the remaining algorithms, all of which are in fact authentication algorithms.
CPA_CY_SYM_HASH_MODE_NESTED	Nested hash. Can be specified for MD5 and the SHA family of hash algorithms.

enum **_CpaCySymHashAlgorithm**

File: cpa_cy_sym.h

Hash algorithms.

This enumeration lists supported hash algorithms.

Enumerator:

CPA_CY_SYM_HASH_NONE	No hash algorithm.
CPA_CY_SYM_HASH_MD5	MD5 algorithm. Supported in all 3 hash modes
CPA_CY_SYM_HASH_SHA1	128 bit SHA algorithm. Supported in all 3 hash modes
CPA_CY_SYM_HASH_SHA224	224 bit SHA algorithm. Supported in all 3 hash modes
CPA_CY_SYM_HASH_SHA256	256 bit SHA algorithm. Supported in all 3 hash modes
CPA_CY_SYM_HASH_SHA384	384 bit SHA algorithm. Supported in all 3 hash modes
CPA_CY_SYM_HASH_SHA512	512 bit SHA algorithm. Supported in all 3 hash modes
CPA_CY_SYM_HASH_AES_XCBC	AES XCBC algorithm. This is only supported in the hash mode CPA_CY_SYM_HASH_MODE_AUTH.
CPA_CY_SYM_HASH_AES_CCM	AES algorithm in CCM mode. This authenticated cipher requires that the hash mode is set to CPA_CY_SYM_HASH_MODE_AUTH. When this hash algorithm is used, the CPA_CY_SYM_CIPHER_AES_CCM element of the CpaCySymCipherAlgorithm enum MUST be used to set up the related CpaCySymCipherSetupData structure in the session context.
CPA_CY_SYM_HASH_AES_GCM	AES algorithm in GCM mode. This authenticated cipher requires that the hash mode is set to CPA_CY_SYM_HASH_MODE_AUTH. When this hash algorithm is used, the CPA_CY_SYM_CIPHER_AES_GCM element of the CpaCySymCipherAlgorithm enum MUST be used to set up the related CpaCySymCipherSetupData structure in the session context.
CPA_CY_SYM_HASH_KASUMI_F9	

	Kasumi algorithm in F9 mode. This is only supported in the hash mode CPA_CY_SYM_HASH_MODE_AUTH.
CPA_CY_SYM_HASH_SNOW3G_UIA2	SNOW3G algorithm in UIA2 mode. This is only supported in the hash mode CPA_CY_SYM_HASH_MODE_AUTH.
CPA_CY_SYM_HASH_AES_CMAC	AES CMAC algorithm. This is only supported in the hash mode CPA_CY_SYM_HASH_MODE_AUTH.
CPA_CY_SYM_HASH_AES_GMAC	AES GMAC algorithm. This is only supported in the hash mode CPA_CY_SYM_HASH_MODE_AUTH. When this hash algorithm is used, the CPA_CY_SYM_CIPHER_AES_GCM element of the CpaCySymCipherAlgorithm enum MUST be used to set up the related CpaCySymCipherSetupData structure in the session context.
CPA_CY_SYM_HASH_AES_CBC_MAC	AES-CBC-MAC algorithm. This is only supported in the hash mode CPA_CY_SYM_HASH_MODE_AUTH. Only 128-bit keys are supported.
CPA_CY_SYM_HASH_ZUC_EIA3	ZUC algorithm in EIA3 mode
CPA_CY_SYM_HASH_SHA3_256	256 bit SHA-3 algorithm. Only CPA_CY_SYM_HASH_MODE_PLAIN and CPA_CY_SYM_HASH_MODE_AUTH are supported, that is, the hash mode CPA_CY_SYM_HASH_MODE_NESTED is not support for this algorithm. Partial requests are not supported, that is, only requests of CPA_CY_SYM_PACKET_TYPE_FULL are supported.

enum **_CpaCySymAlgChainOrder**

File: cpa_cy_sym.h

Algorithm Chaining Operation Ordering

This enum defines the ordering of operations for algorithm chaining.

Enumerator:

CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER

Perform the hash operation followed by the cipher operation. If it is required that the result of the hash (i.e. the digest) is going to be included in the data to be ciphered, then:

- ◊ The digest MUST be placed in the destination buffer at the location corresponding to the end of the data region to be hashed (hashStartSrcOffsetInBytes + messageLenToHashInBytes), i.e. there must be no gaps between the start of the digest and the end of the data region to be hashed.
- ◊ The messageLenToCipherInBytes member of the CpaCySymOpData structure must be equal to the overall length of the plain text, the digest length and any (optional) trailing data that is to be included.
- ◊ The messageLenToCipherInBytes must be a multiple to the block size if a block cipher is being used.

The following is an example of the layout of the buffer before the operation, after the hash, and after the cipher:

```
+-----------------------+---------------+
|         Plaintext     |      Tail     |
+-----------------------+---------------+
<-messageLenToHashInBytes->
```

```
+------------------------+--------+------+
|        Plaintext       | Digest | Tail |
+------------------------+--------+------+
<--------messageLenToCipherInBytes-------->

+---------------------------------------+
|              Cipher Text              |
+---------------------------------------+
```

CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH

Perform the cipher operation followed by the hash operation. The hash operation will be performed on the ciphertext resulting from the cipher operation.

The following is an example of the layout of the buffer before the operation, after the cipher, and after the hash:

```
+--------+---------------------------+---------------+
|  Head  |         Plaintext         |    Tail       |
+--------+---------------------------+---------------+
         <-messageLenToCipherInBytes->

+--------+---------------------------+---------------+
|  Head  |         Ciphertext        |    Tail       |
+--------+---------------------------+---------------+
<------messageLenToHashInBytes------->

+--------+---------------------------+--------+------+
|  Head  |         Ciphertext        | Digest | Tail |
+--------+---------------------------+--------+------+
```

8.12 Function Documentation

CpaStatus cpaCySymSessionCtxGetSize	(	const **CpaInstanceHandle**	*instanceHandle*,
		const **CpaCySymSessionSetupData** *	*pSessionSetupData*,
		Cpa32U *	*pSessionCtxSizeInBytes*
	)		

File: cpa_cy_sym.h

Gets the size required to store a session context.

This function is used by the client to determine the size of the memory it must allocate in order to store the session context. This MUST be called before the client allocates the memory for the session context and before the client calls the **cpaCySymInitSession** function.

For a given implementation of this API, it is safe to assume that **cpaCySymSessionCtxGetSize()** will always return the same size and that the size will not be different for different setup data parameters. However, it should be noted that the size may change: (1) between different implementations of the API (e.g. between software and hardware implementations or between different hardware implementations) (2) between different releases of the same API implementation.

The size returned by this function is the smallest size needed to support all possible combinations of setup data parameters. Some setup data parameter combinations may fit within a smaller session context size. The alternate **cpaCySymSessionCtxGetDynamicSize()** function will return the smallest size needed to fit the provided setup data parameters.

Context:
This is a synchronous function that cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
No.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pSessionSetupData*	Pointer to session setup data which contains parameters which are static for a given cryptographic session such as operation type, mechanisms, and keys for cipher and/or hash operations.
[out]	*pSessionCtxSizeInBytes*	The amount of memory in bytes required to hold the Session Context.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
This is a synchronous function and has no completion callback associated with it.

See also:
CpaCySymSessionSetupData cpaCySymInitSession() cpaCySymSessionCtxGetDynamicSize() cpaCySymPerformOp()

CpaStatus cpaCySymSessionCtxGetDynamicSize	(const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCySymSessionSetupData** *	*pSessionSetupData*,
	Cpa32U *	*pSessionCtxSizeInBytes*
	)	

File: cpa_cy_sym.h

Gets the minimum size required to store a session context.

This function is used by the client to determine the smallest size of the memory it must allocate in order to store the session context. This MUST be called before the client allocates the memory for the session context and before the client calls the **cpaCySymInitSession** function.

This function is an alternate to cpaCySymSessionGetSize(). **cpaCySymSessionCtxGetSize()** will return a fixed size which is the minimum memory size needed to support all possible setup data parameter combinations. **cpaCySymSessionCtxGetDynamicSize()** will return the minimum memory size needed to support the specific session setup data parmeters provided. This size may be different for different setup data parameters.

Context:
This is a synchronous function that cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
No.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pSessionSetupData*	Pointer to session setup data which contains parameters which are static for a given cryptographic session such as operation type, mechanisms, and keys for cipher and/or hash operations.
[out]	*pSessionCtxSizeInBytes*	The amount of memory in bytes required to hold the Session Context.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:

This is a synchronous function and has no completion callback associated with it.

See also:
CpaCySymSessionSetupData cpaCySymInitSession() cpaCySymSessionCtxGetSize() cpaCySymPerformOp()

CpaStatus cpaCySymInitSession (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCySymCbFunc**	*pSymCb*,
	const **CpaCySymSessionSetupData** *	*pSessionSetupData*,
	CpaCySymSessionCtx	*sessionCtx*
)		

File: cpa_cy_sym.h

Initialize a session for symmetric cryptographic API.

This function is used by the client to initialize an asynchronous completion callback function for the symmetric cryptographic operations. Clients MAY register multiple callback functions using this function. The callback function is identified by the combination of userContext, pSymCb and session context (sessionCtx). The session context is the handle to the session and needs to be passed when processing calls. Callbacks on completion of operations within a session are guaranteed to be in the same order they were submitted in.

Context:
This is a synchronous function and it cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
No.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pSymCb*	Pointer to callback function to be registered. Set to NULL if the cpaCySymPerformOp function is required to work in a synchronous manner.
`[in]`	*pSessionSetupData*	Pointer to session setup data which contains parameters which are static for a given cryptographic session such as operation type, mechanisms, and keys for cipher and/or hash operations.
`[out]`	*sessionCtx*	Pointer to the memory allocated by the client to store the session context. This will be initialized with this function. This value needs to be passed to subsequent processing calls.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
This is a synchronous function and has no completion callback associated with it.

See also:
CpaCySymSessionCtx, **CpaCySymCbFunc**, **CpaCySymSessionSetupData**, **cpaCySymRemoveSession()**, **cpaCySymPerformOp()**

CpaStatus cpaCySymRemoveSession (const **CpaInstanceHandle** *instanceHandle*, **CpaCySymSessionCtx** *pSessionCtx*)

File: cpa_cy_sym.h

Remove (delete) a symmetric cryptographic session.

This function will remove a previously initialized session context and the installed callback handler function. Removal will fail if outstanding calls still exist for the initialized session handle. The client needs to retry the remove function at a later time. The memory for the session context MUST not be freed until this call has completed successfully.

Context:
This is a synchronous function that cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
No.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.

`[in,out]`	*pSessionCtx*	Session context to be removed.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized via cpaCyStartInstance function.

Postcondition:

None

Note:

Note that this is a synchronous function and has no completion callback associated with it.

See also:

CpaCySymSessionCtx, **cpaCySymInitSession()**

CpaStatus cpaCySymUpdateSession (	**CpaCySymSessionCtx**	*sessionCtx*,
	const **CpaCySymSessionUpdateData** *	*pSessionUpdateData*
	)	

File: cpa_cy_sym.h

Update a session.

This function is used to update certain parameters of a session, as specified by the CpaCySymSessionUpdateData data structure.

It can be used on sessions created with either the so-called Traditional API (**cpaCySymInitSession**) or the Data Plane API (**cpaCySymDpInitSession**).

In order for this function to operate correctly, two criteria must be met:

- In the case of sessions created with the Traditional API, the session must be stateless, i.e. the field partialsNotRequired of the CpaCySymSessionSetupData data structure must be FALSE. (Sessions created using the Data Plane API are always stateless.)

- There must be no outstanding requests in flight for the session. The application can call the function **cpaCySymSessionInUse** to test for this.

Note that in the case of multi-threaded applications (which are supported using the Traditional API only), this function may fail even if a previous invocation of the function **cpaCySymSessionInUse** indicated that there were no outstanding requests.

Parameters:

`[in]`	*sessionCtx*	Identifies the session to be reset.
`[in]`	*pSessionUpdateData*	Pointer to session data which contains the parameters to be updated.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
This is a synchronous function and has no completion callback associated with it.

```
CpaStatus cpaCySymSessionInUse ( CpaCySymSessionCtx  sessionCtx,
                                 CpaBoolean *        pSessionInUse
                               )
```

File: cpa_cy_sym.h

Indicates whether there are outstanding requests on a given session.

This function is used to test whether there are outstanding requests in flight for a specified session. This may be used before resetting session parameters using the function cpaCySymResetSession. See some additional notes on multi-threaded applications described on that function.

Parameters:

[in]	*sessionCtx*	Identifies the session to be reset.
[out]	*pSessionInUse*	Returns CPA_TRUE if there are outstanding requests on the session, or CPA_FALSE otherwise.

```
CpaStatus cpaCySymPerformOp ( const CpaInstanceHandle  instanceHandle,
                              void *                   pCallbackTag,
                              const CpaCySymOpData *   pOpData,
                              const CpaBufferList *    pSrcBuffer,
                              CpaBufferList *          pDstBuffer,
                              CpaBoolean *             pVerifyResult
                            )
```

File: cpa_cy_sym.h

Perform a symmetric cryptographic operation on an existing session.

Performs a cipher, hash or combined (cipher and hash) operation on the source data buffer using supported symmetric key algorithms and modes.

This function maintains cryptographic state between calls for partial cryptographic operations. If a partial cryptographic operation is being performed, then on a per-session basis, the next part of the multi-part message can be submitted prior to previous parts being completed, the only limitation being that all parts must be performed in sequential order.

If for any reason a client wishes to terminate the partial packet processing on the session (for example if a packet fragment was lost) then the client MUST remove the session.

When using partial packet processing with algorithm chaining, only the cipher state is maintained between calls. The hash state is not be maintained between calls. Instead the hash digest will be generated/verified for each call. If both the cipher state and hash state need to be maintained between calls, algorithm chaining cannot be used.

The following restrictions apply to the length:

- When performing block based operations on a partial packet (excluding the final partial packet), the data that is to be operated on MUST be a multiple of the block size of the algorithm being used. This restriction only applies to the cipher state when using partial packets with algorithm chaining.

- The final block must not be of length zero (0) if the operation being performed is the authentication algorithm **CPA_CY_SYM_HASH_AES_XCBC**. This is because this algorithm requires that the final block be XORed with another value internally. If the length is zero, then the return code **CPA_STATUS_INVALID_PARAM** will be returned.

- The length of the final block must be greater than or equal to 16 bytes when using the **CPA_CY_SYM_CIPHER_AES_XTS** cipher algorithm.

Partial packet processing is supported only when the following conditions are true:

- The cipher, hash or authentication operation is "in place" (that is, pDstBuffer == pSrcBuffer)

- The cipher or hash algorithm is NOT one of Kasumi or SNOW3G

- The cipher mode is NOT F8 mode.

- The instance/implementation supports partial packets as one of its capabilities (see **CpaCySymCapabilitiesInfo**).

The term "in-place" means that the result of the cryptographic operation is written into the source buffer. The term "out-of-place" means that the result of the cryptographic operation is written into the destination buffer. To perform "in-place" processing, set the pDstBuffer parameter to point at the same location as the pSrcBuffer parameter.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:

Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCallbackTag*	Opaque data that will be returned to the client in the callback.
`[in]`	*pOpData*	Pointer to a structure containing request parameters. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[in]`	*pSrcBuffer*	The source buffer. The caller MUST allocate the source buffer and populate it with data. For optimum performance, the data pointed to SHOULD be 8-byte aligned. For block ciphers, the data passed in MUST be a multiple of the relevant block size. i.e. padding WILL NOT be applied to the data. For optimum performance, the buffer should only contain the data region that the cryptographic operation(s) must be performed on. Any additional data in the source buffer may be copied to the destination buffer and this copy may degrade performance.
`[out]`	*pDstBuffer*	The destination buffer. The caller MUST allocate a sufficiently sized destination buffer to hold the data output (including the authentication tag in the case of CCM). Furthermore, the destination buffer must be the same size as the source buffer (i.e. the sum of lengths of the buffers in the buffer list must be the same). This effectively means that the source buffer must in fact be big enough to hold the output data, too. This is because, for out-of-place processing, the data outside the regions in the source buffer on which cryptographic operations are performed are copied into the destination buffer. To perform "in-place" processing set the pDstBuffer parameter in cpaCySymPerformOp function to point at the same location as pSrcBuffer. For optimum performance, the data pointed to SHOULD be 8-byte aligned.
`[out]`	*pVerifyResult*	In synchronous mode, this parameter is returned when the verifyDigest option is set in the CpaCySymSessionSetupData structure. A value of CPA_TRUE indicates that the compare succeeded. A value of CPA_FALSE indicates that the compare failed for an unspecified reason.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resource.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized via cpaCyStartInstance function. A Cryptographic session has been previously setup using the **cpaCySymInitSession** function call.

Postcondition:

None

Note:

When in asynchronous mode, a callback of type CpaCySymCbFunc is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code.

See also:

CpaCySymOpData, **cpaCySymInitSession()**, **cpaCySymRemoveSession()**

CpaStatus CPA_DEPRECATED cpaCySymQueryStats	(	const **CpaInstanceHandle**	*instanceHandle*,
		struct **_CpaCySymStats** *	*pSymStats*
	)		

File: cpa_cy_sym.h

Query symmetric cryptographic statistics for a specific instance.

Deprecated:
As of v1.3 of the cryptographic API, this function has been deprecated, replaced by **cpaCySymQueryStats64()**.

This function will query a specific instance for statistics. The user MUST allocate the CpaCySymStats structure and pass the reference to that into this function call. This function will write the statistic results into the passed in CpaCySymStats structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[out]`	*pSymStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner, i.e. no asynchronous callback will be generated.

See also:
CpaCySymStats

CpaStatus cpaCySymQueryStats64 (	const **CpaInstanceHandle**	*instanceHandle*,
	CpaCySymStats64 *	*pSymStats*
	)	

File: cpa_cy_sym.h

Query symmetric cryptographic statistics (64-bit version) for a specific instance.

This function will query a specific instance for statistics. The user MUST allocate the CpaCySymStats64 structure and pass the reference to that into this function call. This function will write the statistic results into the passed in CpaCySymStats64 structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[out]`	*pSymStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner, i.e. no asynchronous callback will be generated.

See also:
CpaCySymStats64

CpaStatus cpaCySymQueryCapabilities (	const **CpaInstanceHandle**	*instanceHandle*,	
	CpaCySymCapabilitiesInfo *	*pCapInfo*	
)			

File: cpa_cy_sym.h

Returns capabilities of the symmetric API group of a Cryptographic API instance.

This function is used to determine which specific capabilities are supported within the symmetric sub-group of the Cryptographic API.

Context:
The function shall not be called in an interrupt context.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Handle to an instance of this API.
[out]	*pCapInfo*	Pointer to capabilities info structure. All fields in the structure are populated by the API instance.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The instance has been initialized via the **cpaCyStartInstance** function.

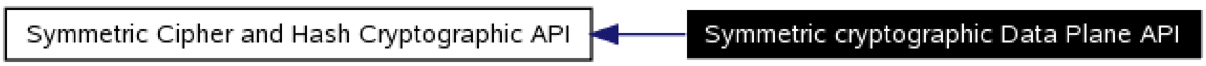
Postcondition:

None

9 Symmetric cryptographic Data Plane API

[Symmetric Cipher and Hash Cryptographic API]

Collaboration diagram for Symmetric cryptographic Data Plane API:



9.1 Detailed Description

File: cpa_cy_sym_dp.h

These data structures and functions specify the Data Plane API for symmetric cipher, hash, and combined cipher and hash operations.

This API is recommended for data plane applications, in which the cost of offload - that is, the cycles consumed by the driver in sending requests to the hardware, and processing responses - needs to be minimized. In particular, use of this API is recommended if the following constraints are acceptable to your application:

- Thread safety is not guaranteed. Each software thread should have access to its own unique instance (CpaInstanceHandle) to avoid contention.
- Polling is used, rather than interrupts (which are expensive). Implementations of this API will provide a function (not defined as part of this API) to read responses from the hardware response queue and dispatch callback functions, as specified on this API.
- Buffers and buffer lists are passed using physical addresses, to avoid virtual to physical address translation costs.
- For GCM and CCM modes of AES, when performing decryption and verification, if verification fails, then the message buffer will NOT be zeroed. (This is a consequence of using physical addresses for the buffers.)
- The ability to enqueue one or more requests without submitting them to the hardware allows for certain costs to be amortized across multiple requests.
- Only asynchronous invocation is supported.
- There is no support for partial packets.
- Implementations may provide certain features as optional at build time, such as atomic counters.
- The "default" instance (**CPA_INSTANCE_HANDLE_SINGLE**) is not supported on this API. The specific handle should be obtained using the instance discovery functions (**cpaCyGetNumInstances**, **cpaCyGetInstances**).

Note:

Performance Trade-Offs Different implementations of this API may have different performance trade-offs; please refer to the documentation for your implementation for details. However, the following concepts informed the definition of this API.

The API distinguishes between *enqueuing* a request and actually *submitting* that request to the cryptographic acceleration engine to be performed. This allows multiple requests to be enqueued (either individually or in batch), and then for all enqueued requests to be submitted in a single operation. The rationale is that in some (especially hardware-based) implementations, the submit operation is expensive; for example, it may incur an MMIO instruction. The API allows this cost to be amortized over a number of requests. The precise number of such requests can be tuned for optimal performance.

Specifically:

- The function **cpaCySymDpEnqueueOp** allows one request to be enqueued, and optionally for that request (and all previously enqueued requests) to be submitted.
- The function **cpaCySymDpEnqueueOpBatch** allows multiple requests to be enqueued, and optionally for those requests (and all previously enqueued requests) to be submitted.
- The function **cpaCySymDpPerformOpNow** enqueues no requests, but submits all previously enqueued requests.

9.2 Data Structures

- struct **_CpaCySymDpOpData**

9.3 Typedefs

- typedef void * **CpaCySymDpSessionCtx**
- typedef **_CpaCySymDpOpData CpaCySymDpOpData**
- typedef void(* **CpaCySymDpCbFunc**)(**CpaCySymDpOpData** *pOpData, **CpaStatus** status, **CpaBoolean** verifyResult)

9.4 Functions

- **CpaStatus cpaCySymDpRegCbFunc** (const **CpaInstanceHandle** instanceHandle, const **CpaCySymDpCbFunc** pSymNewCb)
- **CpaStatus cpaCySymDpSessionCtxGetSize** (const **CpaInstanceHandle** instanceHandle, const **CpaCySymSessionSetupData** *pSessionSetupData, **Cpa32U** *pSessionCtxSizeInBytes)
- **CpaStatus cpaCySymDpSessionCtxGetDynamicSize** (const **CpaInstanceHandle** instanceHandle, const **CpaCySymSessionSetupData** *pSessionSetupData, **Cpa32U** *pSessionCtxSizeInBytes)
- **CpaStatus cpaCySymDpInitSession** (**CpaInstanceHandle** instanceHandle, const **CpaCySymSessionSetupData** *pSessionSetupData, **CpaCySymDpSessionCtx** sessionCtx)
- **CpaStatus cpaCySymDpRemoveSession** (const **CpaInstanceHandle** instanceHandle, **CpaCySymDpSessionCtx** sessionCtx)
- **CpaStatus cpaCySymDpEnqueueOp** (**CpaCySymDpOpData** *pOpData, const **CpaBoolean** performOpNow)
- **CpaStatus cpaCySymDpEnqueueOpBatch** (const **Cpa32U** numberRequests, **CpaCySymDpOpData** *pOpData[], const **CpaBoolean** performOpNow)
- **CpaStatus cpaCySymDpPerformOpNow** (**CpaInstanceHandle** instanceHandle)

9.5 Data Structure Documentation

9.5.1 _CpaCySymDpOpData Struct Reference

9.5.1.1 Detailed Description

File: cpa_cy_sym_dp.h

Operation Data for cryptographic data plane API.

This structure contains data relating to a request to perform symmetric cryptographic processing on one or more data buffers.

The physical memory to which this structure points needs to be at least 8-byte aligned.

All reserved fields SHOULD NOT be written or read by the calling code.

See also:

cpaCySymDpEnqueueOp, **cpaCySymDpEnqueueOpBatch**

9.5.1.2 Data Fields

- **Cpa64U reserved0**
- **Cpa32U cryptoStartSrcOffsetInBytes**
- **Cpa32U messageLenToCipherInBytes**
- **CpaPhysicalAddr iv**
- **Cpa64U reserved1**
- **Cpa32U hashStartSrcOffsetInBytes**
- **Cpa32U messageLenToHashInBytes**
- **CpaPhysicalAddr additionalAuthData**
- **CpaPhysicalAddr digestResult**
- **CpaInstanceHandle instanceHandle**
- **CpaCySymDpSessionCtx sessionCtx**
- **Cpa32U ivLenInBytes**
- **CpaPhysicalAddr srcBuffer**
- **Cpa32U srcBufferLen**
- **CpaPhysicalAddr dstBuffer**
- **Cpa32U dstBufferLen**
- **CpaPhysicalAddr thisPhys**
- **Cpa8U** * **pIv**
- **Cpa8U** * **pAdditionalAuthData**
- void * **pCallbackTag**

9.5.1.3 Field Documentation

Cpa64U _CpaCySymDpOpData::reserved0

Reserved for internal usage.

Cpa32U _CpaCySymDpOpData::cryptoStartSrcOffsetInBytes

Starting point for cipher processing, specified as number of bytes from start of data in the source buffer. The result of the cipher operation will be written back into the buffer starting at this location in the destination buffer.

Cpa32U _CpaCySymDpOpData::messageLenToCipherInBytes

The message length, in bytes, of the source buffer on which the cryptographic operation will be computed. This must be a multiple of the block size if a block cipher is being used. This is also the same as the result length.

Note:

In the case of CCM (**CPA_CY_SYM_HASH_AES_CCM**), this value should not include the length of the padding or the length of the MAC; the driver will compute the actual number of bytes over which the encryption will occur, which will include these values.

For AES-GMAC (**CPA_CY_SYM_HASH_AES_GMAC**), this field should be set to 0.

On some implementations, this length may be limited to a 16-bit value (65535 bytes).

CpaPhysicalAddr _CpaCySymDpOpData::iv

Initialization Vector or Counter. Specifically, this is the physical address of one of the following:

- For block ciphers in CBC mode, or for Kasumi in F8 mode, or for SNOW3G in UEA2 mode, this is the Initialization Vector (IV) value.
- For ARC4, this is reserved for internal usage.
- For block ciphers in CTR mode, this is the counter.
- For GCM mode, this is either the IV (if the length is 96 bits) or J0 (for other sizes), where J0 is as defined by NIST SP800-38D. Regardless of the IV length, a full 16 bytes needs to be allocated.
- For CCM mode, the first byte is reserved, and the nonce should be written starting at &pIv[1] (to allow space for the implementation to write in the flags in the first byte). Note that a full 16 bytes should be allocated, even though the ivLenInBytes field will have a value less than this. The macro **CPA_CY_SYM_CCM_SET_NONCE** may be used here.

Cpa64U _CpaCySymDpOpData::reserved1

Reserved for internal usage.

Cpa32U _CpaCySymDpOpData::hashStartSrcOffsetInBytes

Starting point for hash processing, specified as number of bytes from start of packet in source buffer.

Note:

For CCM and GCM modes of operation, this value in this field is ignored, and the field is reserved for internal usage. The fields **additionalAuthData** and **pAdditionalAuthData** should be set instead.

For AES-GMAC (**CPA_CY_SYM_HASH_AES_GMAC**) mode of operation, this field specifies the start of the AAD data in the source buffer.

Cpa32U _CpaCySymDpOpData::messageLenToHashInBytes

The message length, in bytes, of the source buffer that the hash will be computed on.

Note:

For CCM and GCM modes of operation, this value in this field is ignored, and the field is reserved for internal usage. The fields **additionalAuthData** and **pAdditionalAuthData** should be set instead.

For AES-GMAC (**CPA_CY_SYM_HASH_AES_GMAC**) mode of operation, this field specifies the length of the AAD data in the source buffer.

On some implementations, this length may be limited to a 16-bit value (65535 bytes).

CpaPhysicalAddr _CpaCySymDpOpData::additionalAuthData

Physical address of the Additional Authenticated Data (AAD), which is needed for authenticated cipher mechanisms (CCM and GCM), and to the IV for SNOW3G authentication (**CPA_CY_SYM_HASH_SNOW3G_UIA2**). For other authentication mechanisms, this value is ignored, and the field is reserved for internal usage.

The length of the data pointed to by this field is set up for the session in the **CpaCySymHashAuthModeSetupData** structure as part of the **cpaCySymDpInitSession** function call. This length must not exceed 240 bytes.

If AAD is not used, this address must be set to zero.

Specifically for CCM (**CPA_CY_SYM_HASH_AES_CCM**) and GCM (**CPA_CY_SYM_HASH_AES_GCM**), the caller should be setup as described in the same way as the corresponding field, pAdditionalAuthData, on the "traditional" API (see the **CpaCySymOpData**).

Note:

For AES-GMAC (**CPA_CY_SYM_HASH_AES_GMAC**) mode of operation, this field is not used and should be set to 0. Instead the AAD data should be placed in the source buffer.

CpaPhysicalAddr _CpaCySymDpOpData::digestResult

If the digestIsAppended member of the **CpaCySymSessionSetupData** structure is NOT set then this is the physical address of the location where the digest result should be inserted (in the case of digest generation) or where the purported digest exists (in the case of digest verification).

At session registration time, the client specified the digest result length with the digestResultLenInBytes member of the **CpaCySymHashSetupData** structure. The client must allocate at least digestResultLenInBytes of physically contiguous memory at this location.

For digest generation, the digest result will overwrite any data at this location.

Note:

For GCM (**CPA_CY_SYM_HASH_AES_GCM**), for "digest result" read "authentication tag T".

If the digestIsAppended member of the **CpaCySymSessionSetupData** structure is set then this value is ignored and the digest result is understood to be in the destination buffer for digest generation, and in the source buffer for digest verification. The location of the digest result in this case is immediately following the region over which the digest is computed.

CpaInstanceHandle _CpaCySymDpOpData::instanceHandle

Instance to which the request is to be enqueued.

Note:

A callback function must have been registered on the instance using **cpaCySymDpRegCbFunc**.

CpaCySymDpSessionCtx _CpaCySymDpOpData::sessionCtx

Session context specifying the cryptographic parameters for this request.

Note:

The session must have been created using **cpaCySymDpInitSession**.

Cpa32U _CpaCySymDpOpData::ivLenInBytes

Length of valid IV data pointed to by the pIv parameter.

- For block ciphers in CBC mode, or for Kasumi in F8 mode, or for SNOW3G in UEA2 mode, this is the length of the IV (which must be the same as the block length of the cipher).
- For block ciphers in CTR mode, this is the length of the counter (which must be the same as the block length of the cipher).
- For GCM mode, this is either 12 (for 96-bit IVs) or 16, in which case pIv points to J0.
- For CCM mode, this is the length of the nonce, which can be in the range 7 to 13 inclusive.

CpaPhysicalAddr _CpaCySymDpOpData::srcBuffer

Physical address of the source buffer on which to operate. This is either:

- The location of the data, of length srcBufferLen; or,
- If srcBufferLen has the special value **CPA_DP_BUFLIST**, then srcBuffer contains the location where a **CpaPhysBufferList** is stored. In this case, the CpaPhysBufferList MUST be aligned on an 8-byte boundary.
- For optimum performance, the buffer should only contain the data region that the cryptographic

operation(s) must be performed on. Any additional data in the source buffer may be copied to the destination buffer and this copy may degrade performance.

Cpa32U _CpaCySymDpOpData::srcBufferLen

Length of source buffer, or **CPA_DP_BUFLIST**.

CpaPhysicalAddr _CpaCySymDpOpData::dstBuffer

Physical address of the destination buffer on which to operate. This is either:

- The location of the data, of length srcBufferLen; or,
- If srcBufferLen has the special value **CPA_DP_BUFLIST**, then srcBuffer contains the location where a **CpaPhysBufferList** is stored. In this case, the CpaPhysBufferList MUST be aligned on an 8-byte boundary.

For "in-place" operation, the dstBuffer may be identical to the srcBuffer.

Cpa32U _CpaCySymDpOpData::dstBufferLen

Length of destination buffer, or **CPA_DP_BUFLIST**.

CpaPhysicalAddr _CpaCySymDpOpData::thisPhys

Physical address of this data structure

Cpa8U* **_CpaCySymDpOpData::pIv**

Pointer to (and therefore, the virtual address of) the IV field above. Needed here because the driver in some cases writes to this field, in addition to sending it to the accelerator.

Cpa8U* **_CpaCySymDpOpData::pAdditionalAuthData**

Pointer to (and therefore, the virtual address of) the additionalAuthData field above. Needed here because the driver in some cases writes to this field, in addition to sending it to the accelerator.

void* **_CpaCySymDpOpData::pCallbackTag**

Opaque data that will be returned to the client in the function completion callback.

This opaque data is not used by the implementation of the API, but is simply returned as part of the asynchronous response. It may be used to store information that might be useful when processing the response later.

9.6 Typedef Documentation

typedef void* **CpaCySymDpSessionCtx**

File: cpa_cy_sym_dp.h

Cryptographic component symmetric session context handle for the data plane API.

Handle to a cryptographic data plane session context. The memory for this handle is allocated by the client. The size of the memory that the client needs to allocate is determined by a call to the **cpaCySymDpSessionCtxGetSize** or **cpaCySymDpSessionCtxGetDynamicSize** functions. The session context memory is initialized with a call to the **cpaCySymInitSession** function. This memory MUST not be freed until a call to **cpaCySymDpRemoveSession** has completed successfully.

typedef struct **_CpaCySymDpOpData CpaCySymDpOpData**

File: cpa_cy_sym_dp.h

Operation Data for cryptographic data plane API.

This structure contains data relating to a request to perform symmetric cryptographic processing on one or more data buffers.

The physical memory to which this structure points needs to be at least 8-byte aligned.

All reserved fields SHOULD NOT be written or read by the calling code.

See also:
cpaCySymDpEnqueueOp, **cpaCySymDpEnqueueOpBatch**

typedef void(* **CpaCySymDpCbFunc**)(**CpaCySymDpOpData** *pOpData, **CpaStatus** status, **CpaBoolean** verifyResult)

File: cpa_cy_sym_dp.h

Definition of callback function for cryptographic data plane API.

This is the callback function prototype. The callback function is registered by the application using the **cpaCySymDpRegCbFunc** function call, and called back on completion of asycnhronous requests made via calls to **cpaCySymDpEnqueueOp** or **cpaCySymDpEnqueueOpBatch**.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
No

Parameters:

[in]	*pOpData*	Pointer to the CpaCySymDpOpData object which was supplied as part of the original request.
[in]	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
[in]	*verifyResult*	This parameter is valid when the verifyDigest option is set in the CpaCySymSessionSetupData structure. A value of CPA_TRUE indicates that the compare succeeded. A value of CPA_FALSE indicates that the compare failed.

Returns:
None

Precondition:
Component has been initialized. Callback has been registered with **cpaCySymDpRegCbFunc**.

Postcondition:
None

Note:
None

See also:
cpaCySymDpRegCbFunc

9.7 Function Documentation

CpaStatus cpaCySymDpRegCbFunc (const **CpaInstanceHandle** *instanceHandle*, const **CpaCySymDpCbFunc** *pSymNewCb*)

File: cpa_cy_sym_dp.h

Registration of the operation completion callback function.

This function allows a completion callback function to be registered. The registered callback function is invoked on completion of asycnhronous requests made via calls to **cpaCySymDpEnqueueOp** or **cpaCySymDpEnqueueOpBatch**.

If a callback function was previously registered, it is overwritten.

Context:
This is a synchronous function and it cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
No

Parameters:

[in]	*instanceHandle*	Instance on which the callback function is to be registered.
[in]	*pSymNewCb*	Callback function for this instance.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
CpaCySymDpCbFunc

CpaStatus cpaCySymDpSessionCtxGetSize	(	const **CpaInstanceHandle**	*instanceHandle*,
		const **CpaCySymSessionSetupData** *	*pSessionSetupData*,
		Cpa32U *	*pSessionCtxSizeInBytes*
	)		

File: cpa_cy_sym_dp.h

Gets the size required to store a session context for the data plane API.

This function is used by the client to determine the size of the memory it must allocate in order to store the session context. This MUST be called before the client allocates the memory for the session context and before the client calls the **cpaCySymDpInitSession** function.

For a given implementation of this API, it is safe to assume that **cpaCySymDpSessionCtxGetSize()** will always return the same size and that the size will not be different for different setup data parameters. However, it should be noted that the size may change: (1) between different implementations of the API (e.g. between software and hardware implementations or between different hardware implementations) (2) between different releases of the same API implementation.

The size returned by this function is the smallest size needed to support all possible combinations of setup data parameters. Some setup data parameter combinations may fit within a smaller session context size. The alternate **cpaCySymDpSessionCtxGetDynamicSize()** function will return the smallest size needed to fit the provided setup data parameters.

Context:
This is a synchronous function that cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pSessionSetupData*	Pointer to session setup data which contains parameters which are static for a given cryptographic session such as operation type, mechanisms, and keys for cipher and/or hash operations.
[out]	*pSessionCtxSizeInBytes*	The amount of memory in bytes required to hold the Session Context.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized.

Postcondition:

None

Note:

This is a synchronous function and has no completion callback associated with it.

See also:

CpaCySymSessionSetupData cpaCySymDpSessionCtxGetDynamicSize() cpaCySymDpInitSession()

CpaStatus cpaCySymDpSessionCtxGetDynamicSize	(const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCySymSessionSetupData** *	*pSessionSetupData*,
	Cpa32U *	*pSessionCtxSizeInBytes*
	)	

File: cpa_cy_sym_dp.h

Gets the minimum size required to store a session context for the data plane API.

This function is used by the client to determine the smallest size of the memory it must allocate in order to store the session context. This MUST be called before the client allocates the memory for the session context and before the client calls the **cpaCySymDpInitSession** function.

This function is an alternate to cpaCySymDpSessionGetSize(). **cpaCySymDpSessionCtxGetSize()** will return a fixed size which is the minimum memory size needed to support all possible setup data parameter combinations. **cpaCySymDpSessionCtxGetDynamicSize()** will return the minimum memory size needed to support the specific session setup data parmeters provided. This size may be different for different setup data parameters.

Context:

This is a synchronous function that cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:

None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pSessionSetupData*	Pointer to session setup data which contains parameters which are static for a given cryptographic session such as operation type, mechanisms, and keys for cipher and/or hash operations.
[out]	*pSessionCtxSizeInBytes*	The amount of memory in bytes required to hold the Session Context.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized.

Postcondition:
None

Note:
This is a synchronous function and has no completion callback associated with it.

See also:
CpaCySymSessionSetupData cpaCySymDpSessionCtxGetSize() cpaCySymDpInitSession()

CpaStatus cpaCySymDpInitSession (	**CpaInstanceHandle**	*instanceHandle*,
	const **CpaCySymSessionSetupData** *	*pSessionSetupData*,
	CpaCySymDpSessionCtx	*sessionCtx*
)		

File: cpa_cy_sym_dp.h

Initialize a session for the symmetric cryptographic data plane API.

This function is used by the client to initialize an asynchronous session context for symmetric cryptographic data plane operations. The returned session context is the handle to the session and needs to be passed when requesting cryptographic operations to be performed.

Only sessions created using this function may be used when invoking functions on this API

The session can be removed using **cpaCySymDpRemoveSession**.

Context:
This is a synchronous function and it cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
No

Parameters:

[in]	*instanceHandle*	Instance to which the requests will be submitted.
[in]	*pSessionSetupData*	Pointer to session setup data which contains parameters that are static for a given cryptographic session such as operation type, algorithm, and keys for cipher and/or hash operations.
[out]	*sessionCtx*	Pointer to the memory allocated by the client to store the session context. This memory must be physically contiguous, and its length (in bytes) must be at least as big as specified by a call to **cpaCySymDpSessionCtxGetSize**. This memory will be initialized with this function. This value needs to be passed to subsequent processing calls.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized.

Postcondition:
None

Note:
This is a synchronous function and has no completion callback associated with it.

See also:
cpaCySymDpSessionCtxGetSize, **cpaCySymDpRemoveSession**

CpaStatus cpaCySymDpRemoveSession (const **CpaInstanceHandle** *instanceHandle*,
CpaCySymDpSessionCtx *sessionCtx*
)

File: cpa_cy_sym_dp.h

Remove (delete) a symmetric cryptographic session for the data plane API.

This function will remove a previously initialized session context and the installed callback handler function. Removal will fail if outstanding calls still exist for the initialized session handle. The client needs to retry the remove function at a later time. The memory for the session context MUST not be freed until this call has completed successfully.

Context:
This is a synchronous function that cannot sleep. It can be executed in a context that does not permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in,out]`	*sessionCtx*	Session context to be removed.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized.

Postcondition:
None

Note:
Note that this is a synchronous function and has no completion callback associated with it.

See also:
CpaCySymDpSessionCtx, **cpaCySymDpInitSession()**

CpaStatus cpaCySymDpEnqueueOp (	**CpaCySymDpOpData** *	*pOpData*,	
	const **CpaBoolean**	*performOpNow*	
)			

File: cpa_cy_sym_dp.h

Enqueue a single symmetric cryptographic request.

This function enqueues a single request to perform a cipher, hash or combined (cipher and hash) operation. Optionally, the request is also submitted to the cryptographic engine to be performed.

See note about performance trade-offs on the **Symmetric cryptographic Data Plane API** API.

The function is asynchronous; control is returned to the user once the request has been submitted. On completion of the request, the application may poll for responses, which will cause a callback function (registered via **cpaCySymDpRegCbFunc**) to be invoked. Callbacks within a session are guaranteed to be in the same order in which they were submitted.

The following restrictions apply to the pOpData parameter:

- The memory MUST be aligned on an 8-byte boundary.
- The structure MUST reside in physically contiguous memory.
- The reserved fields of the structure SHOULD NOT be written or read by the calling code.

Context:
This function will not sleep, and hence can be executed in a context that does not permit sleeping.

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
No

Parameters:

`[in]`	*pOpData*	Pointer to a structure containing the request parameters. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback, which was registered on the instance via **cpaCySymDpRegCbFunc**. See the above Description for restrictions that apply to this parameter.
`[in]`	*performOpNow*	Flag to specify whether the operation should be performed immediately (CPA_TRUE), or simply enqueued to be performed later (CPA_FALSE). In the latter case, the request is submitted to be performed either by calling this function again with this flag set to CPA_TRUE, or by invoking the function **cpaCySymDpPerformOpNow**.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.

CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The session identified by pOpData->sessionCtx was setup using **cpaCySymDpInitSession**. The instance identified by pOpData->instanceHandle has had a callback function registered via **cpaCySymDpRegCbFunc**.

Postcondition:

None

Note:

A callback of type **CpaCySymDpCbFunc** is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code.

See also:

cpaCySymDpInitSession, **cpaCySymDpPerformOpNow**

CpaStatus cpaCySymDpEnqueueOpBatch (	const **Cpa32U**	*numberRequests*,
	CpaCySymDpOpData *	*pOpData*[],
	const **CpaBoolean**	*performOpNow*
)		

File: cpa_cy_sym_dp.h

Enqueue multiple requests to the symmetric cryptographic data plane API.

This function enqueues multiple requests to perform cipher, hash or combined (cipher and hash) operations.

See note about performance trade-offs on the **Symmetric cryptographic Data Plane API** API.

The function is asynchronous; control is returned to the user once the request has been submitted. On completion of the request, the application may poll for responses, which will cause a callback function (registered via **cpaCySymDpRegCbFunc**) to be invoked. Separate callbacks will be invoked for each request. Callbacks within a session are guaranteed to be in the same order in which they were submitted.

The following restrictions apply to each element of the pOpData array:

- The memory MUST be aligned on an 8-byte boundary.
- The structure MUST reside in physically contiguous memory.
- The reserved fields of the structure SHOULD NOT be written or read by the calling code.

Context:

This function will not sleep, and hence can be executed in a context that does not permit sleeping.

Assumptions:

Client MUST allocate the request parameters to 8 byte alignment. Reserved elements of the CpaCySymDpOpData structure MUST be 0. The CpaCySymDpOpData structure MUST reside in physically contiguous memory.

Side-Effects:

None

Blocking:

No

Reentrant:
No

Thread-safe:
No

Parameters:

[in]	*numberRequests*	The number of requests in the array of CpaCySymDpOpData structures.
[in]	*pOpData*	An array of pointers to CpaCySymDpOpData structures. Each of the CpaCySymDpOpData structure contains the request parameters for that request. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback, which was registered on the instance via **cpaCySymDpRegCbFunc**. See the above Description for restrictions that apply to this parameter.
[in]	*performOpNow*	Flag to specify whether the operation should be performed immediately (CPA_TRUE), or simply enqueued to be performed later (CPA_FALSE). In the latter case, the request is submitted to be performed either by calling this function again with this flag set to CPA_TRUE, or by invoking the function **cpaCySymDpPerformOpNow**.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The session identified by pOpData[i]->sessionCtx was setup using **cpaCySymDpInitSession**. The instance identified by pOpData->instanceHandle[i] has had a callback function registered via **cpaCySymDpRegCbFunc**.

Postcondition:
None

Note:
Multiple callbacks of type **CpaCySymDpCbFunc** are generated in response to this function call (one per request). Any errors generated during processing are reported as part of the callback status code.

See also:
cpaCySymDpInitSession, **cpaCySymDpEnqueueOp**

CpaStatus cpaCySymDpPerformOpNow (**CpaInstanceHandle** *instanceHandle*)

File: cpa_cy_sym_dp.h

Submit any previously enqueued requests to be performed now on the symmetric cryptographic data plane API.

If any requests/operations were enqueued via calls to **cpaCySymDpEnqueueOp** and/or **cpaCySymDpEnqueueOpBatch**, but with the flag performOpNow set to **CPA_FALSE**, then these operations will now be submitted to the accelerator to be performed.

See note about performance trade-offs on the **Symmetric cryptographic Data Plane API** API.

Context:
Will not sleep. It can be executed in a context that does not permit sleeping.

Side-Effects:
None

Blocking:
No

Reentrant:
No

Thread-safe:
No

Parameters:
`[in]` *instanceHandle* Instance to which the requests will be submitted.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized. A cryptographic session has been previously setup using the **cpaCySymDpInitSession** function call.

Postcondition:
None

See also:
cpaCySymDpEnqueueOp, **cpaCySymDpEnqueueOpBatch**

10 Cryptographic Key and Mask Generation API

[Cryptographic API]

Collaboration diagram for Cryptographic Key and Mask Generation API:

10.1 Detailed Description

File: cpa_cy_key.h

These functions specify the API for key and mask generation operations.

10.2 Data Structures

- struct **_CpaCyKeyGenSslOpData**
- struct **_CpaCyKeyGenTlsOpData**
- struct **_CpaCyKeyGenMgfOpData**
- struct **_CpaCyKeyGenMgfOpDataExt**
- struct **_CpaCyKeyGenStats**
- struct **_CpaCyKeyGenStats64**

10.3 Defines

- #define **CPA_CY_KEY_GEN_SSL_TLS_RANDOM_LEN_IN_BYTES**

10.4 Typedefs

- typedef enum **_CpaCyKeySslOp CpaCyKeySslOp**
- typedef **_CpaCyKeyGenSslOpData CpaCyKeyGenSslOpData**
- typedef enum **_CpaCyKeyTlsOp CpaCyKeyTlsOp**
- typedef **_CpaCyKeyGenTlsOpData CpaCyKeyGenTlsOpData**
- typedef **_CpaCyKeyGenMgfOpData CpaCyKeyGenMgfOpData**
- typedef **_CpaCyKeyGenMgfOpDataExt CpaCyKeyGenMgfOpDataExt**
- typedef **_CpaCyKeyGenStats CPA_DEPRECATED**
- typedef **_CpaCyKeyGenStats64 CpaCyKeyGenStats64**

10.5 Enumerations

- enum **_CpaCyKeySslOp** {
  CPA_CY_KEY_SSL_OP_MASTER_SECRET_DERIVE,
  CPA_CY_KEY_SSL_OP_KEY_MATERIAL_DERIVE,
  CPA_CY_KEY_SSL_OP_USER_DEFINED
  }
- enum **_CpaCyKeyTlsOp** {
  CPA_CY_KEY_TLS_OP_MASTER_SECRET_DERIVE,
  CPA_CY_KEY_TLS_OP_KEY_MATERIAL_DERIVE,
  CPA_CY_KEY_TLS_OP_CLIENT_FINISHED_DERIVE,
  CPA_CY_KEY_TLS_OP_SERVER_FINISHED_DERIVE,

CPA_CY_KEY_TLS_OP_USER_DEFINED
}

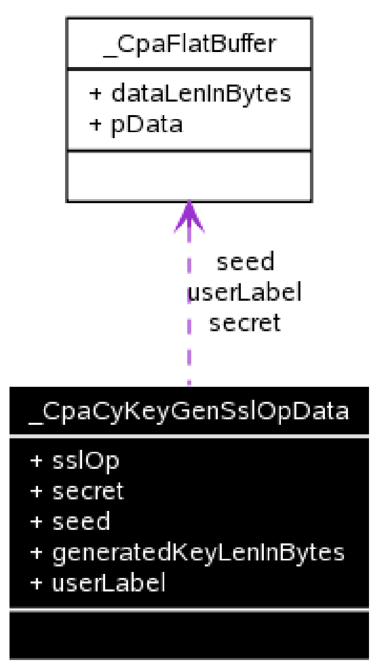
10.6 Functions

- **CpaStatus cpaCyKeyGenSsl** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pKeyGenCb, void *pCallbackTag, const **CpaCyKeyGenSslOpData** *pKeyGenSslOpData, **CpaFlatBuffer** *pGeneratedKeyBuffer)
- **CpaStatus cpaCyKeyGenTls** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pKeyGenCb, void *pCallbackTag, const **CpaCyKeyGenTlsOpData** *pKeyGenTlsOpData, **CpaFlatBuffer** *pGeneratedKeyBuffer)
- **CpaStatus cpaCyKeyGenTls2** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pKeyGenCb, void *pCallbackTag, const **CpaCyKeyGenTlsOpData** *pKeyGenTlsOpData, **CpaCySymHashAlgorithm** hashAlgorithm, **CpaFlatBuffer** *pGeneratedKeyBuffer)
- **CpaStatus cpaCyKeyGenMgf** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pKeyGenCb, void *pCallbackTag, const **CpaCyKeyGenMgfOpData** *pKeyGenMgfOpData, **CpaFlatBuffer** *pGeneratedMaskBuffer)
- **CpaStatus cpaCyKeyGenMgfExt** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pKeyGenCb, void *pCallbackTag, const **CpaCyKeyGenMgfOpDataExt** *pKeyGenMgfOpDataExt, **CpaFlatBuffer** *pGeneratedMaskBuffer)
- **CpaStatus CPA_DEPRECATED cpaCyKeyGenQueryStats** (const **CpaInstanceHandle** instanceHandle, struct **_CpaCyKeyGenStats** *pKeyGenStats)
- **CpaStatus cpaCyKeyGenQueryStats64** (const **CpaInstanceHandle** instanceHandle, **CpaCyKeyGenStats64** *pKeyGenStats)

10.7 Data Structure Documentation

10.7.1 _CpaCyKeyGenSslOpData Struct Reference

Collaboration diagram for _CpaCyKeyGenSslOpData:



10.7.1 _CpaCyKeyGenSslOpData Struct Reference

10.7.1.1 Detailed Description

File: cpa_cy_key.h

SSL data for key generation functions

This structure contains data for use in key generation operations for SSL. For specific SSL key generation operations, the structure fields MUST be set as follows:

SSL Master-Secret Derivation:

sslOp = CPA_CY_KEY_SSL_OP_MASTER_SECRET_DERIVE
secret = pre-master secret key
seed = client_random + server_random
userLabel = NULL

SSL Key-Material Derivation:

sslOp = CPA_CY_KEY_SSL_OP_KEY_MATERIAL_DERIVE
secret = master secret key
seed = server_random + client_random
userLabel = NULL

Note that the client/server random order is reversed from that used for master-secret derivation.

Note:

Each of the client and server random numbers need to be of length CPA_CY_KEY_GEN_SSL_TLS_RANDOM_LEN_IN_BYTES.

In each of the above descriptions, + indicates concatenation.

The label used is predetermined by the SSL operation in line with the SSL 3.0 specification, and can be overridden by using a user defined operation CPA_CY_KEY_SSL_OP_USER_DEFINED and associated userLabel.

10.7.1.2 Data Fields

- **CpaCyKeySslOp sslOp**
- **CpaFlatBuffer secret**
- **CpaFlatBuffer seed**
- **Cpa32U generatedKeyLenInBytes**
- **CpaFlatBuffer userLabel**

10.7.1.3 Field Documentation

CpaCyKeySslOp _CpaCyKeyGenSslOpData::sslOp

Indicate the SSL operation to be performed

CpaFlatBuffer _CpaCyKeyGenSslOpData::secret

Flat buffer containing a pointer to either the master or pre-master secret key. The length field indicates the length of the secret key in bytes. Implementation-specific limits may apply to this length.

CpaFlatBuffer _CpaCyKeyGenSslOpData::seed

Flat buffer containing a pointer to the seed data. Implementation-specific limits may apply to this length.

Cpa32U _CpaCyKeyGenSslOpData::generatedKeyLenInBytes

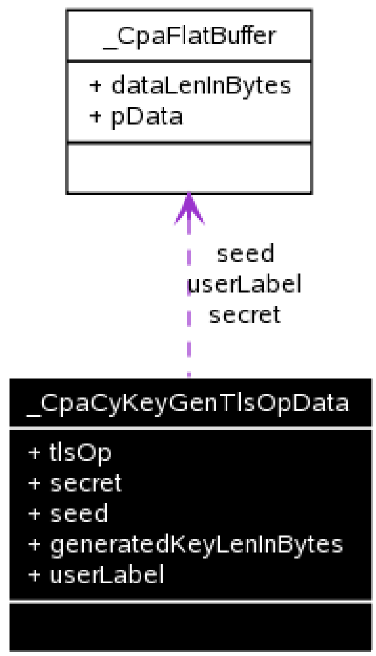
The requested length of the generated key in bytes. Implementation-specific limits may apply to this length.

CpaFlatBuffer _CpaCyKeyGenSslOpData::userLabel

Optional flat buffer containing a pointer to a user defined label. The length field indicates the length of the label in bytes. To use this field, the sslOp must be CPA_CY_KEY_SSL_OP_USER_DEFINED, otherwise it is ignored and can be set to NULL. Implementation-specific limits may apply to this length.

10.7.2 _CpaCyKeyGenTlsOpData Struct Reference

Collaboration diagram for _CpaCyKeyGenTlsOpData:



10.7.2.1 Detailed Description

File: cpa_cy_key.h

TLS data for key generation functions

This structure contains data for use in key generation operations for TLS. For specific TLS key generation operations, the structure fields MUST be set as follows:

TLS Master-Secret Derivation:

tlsOp = CPA_CY_KEY_TLS_OP_MASTER_SECRET_DERIVE
secret = pre-master secret key
seed = client_random + server_random
userLabel = NULL

TLS Key-Material Derivation:

tlsOp = CPA_CY_KEY_TLS_OP_KEY_MATERIAL_DERIVE
secret = master secret key
seed = server_random + client_random
userLabel = NULL

Note that the client/server random order is reversed from that used for Master-Secret Derivation.

TLS Client finished/Server finished tag Derivation:

tlsOp = CPA_CY_KEY_TLS_OP_CLIENT_FINISHED_DERIVE (client)
or CPA_CY_KEY_TLS_OP_SERVER_FINISHED_DERIVE (server)
secret = master secret key
seed = MD5(handshake_messages) + SHA-1(handshake_messages)
userLabel = NULL

Note:

Each of the client and server random seeds need to be of length CPA_CY_KEY_GEN_SSL_TLS_RANDOM_LEN_IN_BYTES.

In each of the above descriptions, + indicates concatenation.

The label used is predetermined by the TLS operation in line with the TLS specifications, and can be overridden by using a user defined operation CPA_CY_KEY_TLS_OP_USER_DEFINED and associated userLabel.

10.7.2.2 Data Fields

- **CpaCyKeyTlsOp tlsOp**
- **CpaFlatBuffer secret**
- **CpaFlatBuffer seed**
- **Cpa32U generatedKeyLenInBytes**
- **CpaFlatBuffer userLabel**

10.7.2.3 Field Documentation

CpaCyKeyTlsOp _CpaCyKeyGenTlsOpData::tlsOp

TLS operation to be performed

CpaFlatBuffer _CpaCyKeyGenTlsOpData::secret

Flat buffer containing a pointer to either the master or pre-master secret key. The length field indicates the length of the secret in bytes. Implementation-specific limits may apply to this length.

CpaFlatBuffer _CpaCyKeyGenTlsOpData::seed

Flat buffer containing a pointer to the seed data. Implementation-specific limits may apply to this length.

Cpa32U _CpaCyKeyGenTlsOpData::generatedKeyLenInBytes

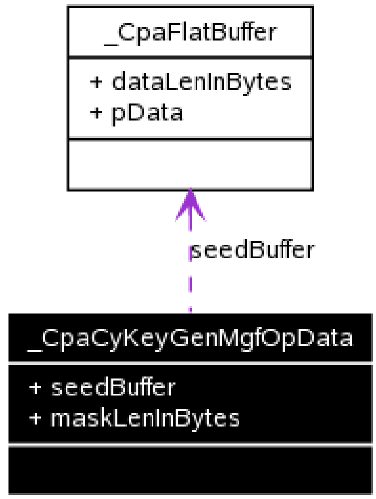
The requested length of the generated key in bytes. Implementation-specific limits may apply to this length.

CpaFlatBuffer _CpaCyKeyGenTlsOpData::userLabel

Optional flat buffer containing a pointer to a user defined label. The length field indicates the length of the label in bytes. To use this field, the tlsOp must be CPA_CY_KEY_TLS_OP_USER_DEFINED, otherwise it is ignored and can be set to NULL. Implementation-specific limits may apply to this length.

10.7.3 _CpaCyKeyGenMgfOpData Struct Reference

Collaboration diagram for _CpaCyKeyGenMgfOpData:



10.7.3.1 Detailed Description

File: cpa_cy_key.h

Key Generation Mask Generation Function (MGF) Data

This structure contains data relating to Mask Generation Function key generation operations.

Note:
The default hash algorithm used by the MGF is SHA-1. If a different hash algorithm is preferred, then see the extended version of this structure, **CpaCyKeyGenMgfOpDataExt**.

See also:
cpaCyKeyGenMgf

10.7.3.2 Data Fields

- **CpaFlatBuffer seedBuffer**
- **Cpa32U maskLenInBytes**

10.7.3.3 Field Documentation

CpaFlatBuffer _CpaCyKeyGenMgfOpData::seedBuffer

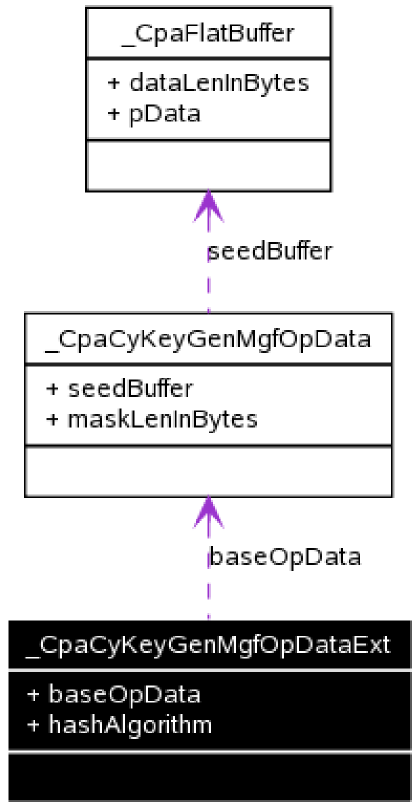
Caller MUST allocate a buffer and populate with the input seed data. For optimal performance the start of the seed SHOULD be allocated on an 8-byte boundary. The length field represents the seed length in bytes. Implementation-specific limits may apply to this length.

Cpa32U _CpaCyKeyGenMgfOpData::maskLenInBytes

The requested length of the generated mask in bytes. Implementation-specific limits may apply to this length.

10.7.4 _CpaCyKeyGenMgfOpDataExt Struct Reference

Collaboration diagram for _CpaCyKeyGenMgfOpDataExt:



10.7.4.1 Detailed Description

File: cpa_cy_key.h

Extension to the original Key Generation Mask Generation Function (MGF) Data

This structure is an extension to the original MGF data structure. The extension allows the hash function to be specified.

Note:

This structure is separate from the base **CpaCyKeyGenMgfOpData** structure in order to retain backwards compatibility with the original version of the API.

See also:

cpaCyKeyGenMgfExt

10.7.4.2 Data Fields

- **CpaCyKeyGenMgfOpData baseOpData**
- **CpaCySymHashAlgorithm hashAlgorithm**

10.7.4.3 Field Documentation

CpaCyKeyGenMgfOpData _CpaCyKeyGenMgfOpDataExt::baseOpData

"Base" operational data for MGF generation

CpaCySymHashAlgorithm _CpaCyKeyGenMgfOpDataExt::hashAlgorithm

Specifies the hash algorithm to be used by the Mask Generation Function

10.7.5 _CpaCyKeyGenStats Struct Reference

10.7.5.1 Detailed Description

File: cpa_cy_key.h

Key Generation Statistics.

Deprecated:

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyKeyGenStats64**.

This structure contains statistics on the key and mask generation operations. Statistics are set to zero when the component is initialized, and are collected per instance.

10.7.5.2 Data Fields

- **Cpa32U numSslKeyGenRequests**
- **Cpa32U numSslKeyGenRequestErrors**
- **Cpa32U numSslKeyGenCompleted**
- **Cpa32U numSslKeyGenCompletedErrors**
- **Cpa32U numTlsKeyGenRequests**
- **Cpa32U numTlsKeyGenRequestErrors**
- **Cpa32U numTlsKeyGenCompleted**
- **Cpa32U numTlsKeyGenCompletedErrors**
- **Cpa32U numMgfKeyGenRequests**
- **Cpa32U numMgfKeyGenRequestErrors**
- **Cpa32U numMgfKeyGenCompleted**
- **Cpa32U numMgfKeyGenCompletedErrors**

10.7.5.3 Field Documentation

Cpa32U _CpaCyKeyGenStats::numSslKeyGenRequests

Total number of successful SSL key generation requests.

Cpa32U _CpaCyKeyGenStats::numSslKeyGenRequestErrors

Total number of SSL key generation requests that had an error and could not be processed.

Cpa32U _CpaCyKeyGenStats::numSslKeyGenCompleted

Total number of SSL key generation operations that completed successfully.

Cpa32U _CpaCyKeyGenStats::numSslKeyGenCompletedErrors

Total number of SSL key generation operations that could not be completed successfully due to errors.

Cpa32U _CpaCyKeyGenStats::numTlsKeyGenRequests

Total number of successful TLS key generation requests.

Cpa32U _CpaCyKeyGenStats::numTlsKeyGenRequestErrors

Total number of TLS key generation requests that had an error and could not be processed.

Cpa32U _CpaCyKeyGenStats::numTlsKeyGenCompleted

Total number of TLS key generation operations that completed successfully.

Cpa32U _CpaCyKeyGenStats::numTlsKeyGenCompletedErrors

Total number of TLS key generation operations that could not be completed successfully due to errors.

Cpa32U _CpaCyKeyGenStats::numMgfKeyGenRequests

Total number of successful MGF key generation requests (including "extended" MGF requests).

Cpa32U _CpaCyKeyGenStats::numMgfKeyGenRequestErrors

Total number of MGF key generation requests that had an error and could not be processed.

Cpa32U _CpaCyKeyGenStats::numMgfKeyGenCompleted

Total number of MGF key generation operations that completed successfully.

Cpa32U _CpaCyKeyGenStats::numMgfKeyGenCompletedErrors

Total number of MGF key generation operations that could not be completed successfully due to errors.

10.7.6 _CpaCyKeyGenStats64 Struct Reference

10.7.6.1 Detailed Description

File: cpa_cy_key.h

Key Generation Statistics (64-bit version).

This structure contains the 64-bit version of the statistics on the key and mask generation operations. Statistics are set to zero when the component is initialized, and are collected per instance.

10.7.6.2 Data Fields

- **Cpa64U numSslKeyGenRequests**
- **Cpa64U numSslKeyGenRequestErrors**
- **Cpa64U numSslKeyGenCompleted**
- **Cpa64U numSslKeyGenCompletedErrors**
- **Cpa64U numTlsKeyGenRequests**
- **Cpa64U numTlsKeyGenRequestErrors**
- **Cpa64U numTlsKeyGenCompleted**
- **Cpa64U numTlsKeyGenCompletedErrors**
- **Cpa64U numMgfKeyGenRequests**
- **Cpa64U numMgfKeyGenRequestErrors**
- **Cpa64U numMgfKeyGenCompleted**
- **Cpa64U numMgfKeyGenCompletedErrors**

10.7.6.3 Field Documentation

Cpa64U _CpaCyKeyGenStats64::numSslKeyGenRequests

Total number of successful SSL key generation requests.

Cpa64U _CpaCyKeyGenStats64::numSslKeyGenRequestErrors

Total number of SSL key generation requests that had an error and could not be processed.

Cpa64U _CpaCyKeyGenStats64::numSslKeyGenCompleted

Total number of SSL key generation operations that completed successfully.

Cpa64U _CpaCyKeyGenStats64::numSslKeyGenCompletedErrors

Total number of SSL key generation operations that could not be completed successfully due to errors.

Cpa64U _CpaCyKeyGenStats64::numTlsKeyGenRequests

Total number of successful TLS key generation requests.

Cpa64U _CpaCyKeyGenStats64::numTlsKeyGenRequestErrors

Total number of TLS key generation requests that had an error and could not be processed.

Cpa64U _CpaCyKeyGenStats64::numTlsKeyGenCompleted

Total number of TLS key generation operations that completed successfully.

Cpa64U _CpaCyKeyGenStats64::numTlsKeyGenCompletedErrors

Total number of TLS key generation operations that could not be completed successfully due to errors.

Cpa64U _CpaCyKeyGenStats64::numMgfKeyGenRequests

Total number of successful MGF key generation requests (including "extended" MGF requests).

Cpa64U _CpaCyKeyGenStats64::numMgfKeyGenRequestErrors

Total number of MGF key generation requests that had an error and could not be processed.

Cpa64U _CpaCyKeyGenStats64::numMgfKeyGenCompleted

Total number of MGF key generation operations that completed successfully.

Cpa64U _CpaCyKeyGenStats64::numMgfKeyGenCompletedErrors

Total number of MGF key generation operations that could not be completed successfully due to errors.

10.8 Define Documentation

#define CPA_CY_KEY_GEN_SSL_TLS_RANDOM_LEN_IN_BYTES

File: cpa_cy_key.h

SSL or TLS key generation random number length.

Defines the permitted SSL or TLS random number length in bytes that may be used with the functions **cpaCyKeyGenSsl** and **cpaCyKeyGenTls**. This is the length of the client or server random number values.

10.9 Typedef Documentation

typedef enum **_CpaCyKeySslOp CpaCyKeySslOp**

File: cpa_cy_key.h

SSL Operation Types

Enumeration of the different SSL operations that can be specified in the struct **CpaCyKeyGenSslOpData**. It identifies the label.

typedef struct **_CpaCyKeyGenSslOpData CpaCyKeyGenSslOpData**

File: cpa_cy_key.h

SSL data for key generation functions

This structure contains data for use in key generation operations for SSL. For specific SSL key generation operations, the structure fields MUST be set as follows:

SSL Master-Secret Derivation:

sslOp = CPA_CY_KEY_SSL_OP_MASTER_SECRET_DERIVE
secret = pre-master secret key
seed = client_random + server_random
userLabel = NULL

SSL Key-Material Derivation:

sslOp = CPA_CY_KEY_SSL_OP_KEY_MATERIAL_DERIVE
secret = master secret key
seed = server_random + client_random
userLabel = NULL

Note that the client/server random order is reversed from that used for master-secret derivation.

Note:

Each of the client and server random numbers need to be of length CPA_CY_KEY_GEN_SSL_TLS_RANDOM_LEN_IN_BYTES.

In each of the above descriptions, + indicates concatenation.

The label used is predetermined by the SSL operation in line with the SSL 3.0 specification, and can be overridden by using a user defined operation CPA_CY_KEY_SSL_OP_USER_DEFINED and associated userLabel.

typedef enum **_CpaCyKeyTlsOp CpaCyKeyTlsOp**

File: cpa_cy_key.h

TLS Operation Types

Enumeration of the different TLS operations that can be specified in the CpaCyKeyGenTlsOpData. It identifies the label.

The functions **cpaCyKeyGenTls** and **cpaCyKeyGenTls2** accelerate the TLS PRF, which is defined as part of RFC2246 (TLS v1.0), RFC4346 (TLS v1.1), and RFC5246 (TLS v1.2). One of the inputs to these functions is a label. This enumerated type defines values that correspond to some of the required labels. However, for some of the operations/labels required by these RFCs, no values are specified.

In such cases, a user-defined value must be provided. The client should use the enum value **CPA_CY_KEY_TLS_OP_USER_DEFINED**, and pass the label using the userLabel field of the **CpaCyKeyGenTlsOpData** data structure.

typedef struct **_CpaCyKeyGenTlsOpData CpaCyKeyGenTlsOpData**

File: cpa_cy_key.h

TLS data for key generation functions

This structure contains data for use in key generation operations for TLS. For specific TLS key generation operations, the structure fields MUST be set as follows:

TLS Master-Secret Derivation:

tlsOp = CPA_CY_KEY_TLS_OP_MASTER_SECRET_DERIVE
secret = pre-master secret key
seed = client_random + server_random
userLabel = NULL

TLS Key-Material Derivation:

tlsOp = CPA_CY_KEY_TLS_OP_KEY_MATERIAL_DERIVE
secret = master secret key
seed = server_random + client_random
userLabel = NULL

Note that the client/server random order is reversed from that used for Master-Secret Derivation.

TLS Client finished/Server finished tag Derivation:

tlsOp = CPA_CY_KEY_TLS_OP_CLIENT_FINISHED_DERIVE (client)
or CPA_CY_KEY_TLS_OP_SERVER_FINISHED_DERIVE (server)
secret = master secret key
seed = MD5(handshake_messages) + SHA-1(handshake_messages)
userLabel = NULL

Note:

Each of the client and server random seeds need to be of length CPA_CY_KEY_GEN_SSL_TLS_RANDOM_LEN_IN_BYTES.

In each of the above descriptions, + indicates concatenation.

The label used is predetermined by the TLS operation in line with the TLS specifications, and can be overridden by using a user defined operation CPA_CY_KEY_TLS_OP_USER_DEFINED and associated userLabel.

typedef struct **_CpaCyKeyGenMgfOpData CpaCyKeyGenMgfOpData**

File: cpa_cy_key.h

Key Generation Mask Generation Function (MGF) Data

This structure contains data relating to Mask Generation Function key generation operations.

Note:
The default hash algorithm used by the MGF is SHA-1. If a different hash algorithm is preferred, then see the extended version of this structure, **CpaCyKeyGenMgfOpDataExt**.

See also:
cpaCyKeyGenMgf

typedef struct **_CpaCyKeyGenMgfOpDataExt CpaCyKeyGenMgfOpDataExt**

File: cpa_cy_key.h

Extension to the original Key Generation Mask Generation Function (MGF) Data

This structure is an extension to the original MGF data structure. The extension allows the hash function to be specified.

Note:
This structure is separate from the base **CpaCyKeyGenMgfOpData** structure in order to retain backwards compatibility with the original version of the API.

See also:
cpaCyKeyGenMgfExt

typedef struct **_CpaCyKeyGenStats CPA_DEPRECATED**

File: cpa_cy_key.h

Key Generation Statistics.

Deprecated:
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyKeyGenStats64**.

This structure contains statistics on the key and mask generation operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef struct **_CpaCyKeyGenStats64 CpaCyKeyGenStats64**

File: cpa_cy_key.h

Key Generation Statistics (64-bit version).

This structure contains the 64-bit version of the statistics on the key and mask generation operations. Statistics are set to zero when the component is initialized, and are collected per instance.

10.10 Enumeration Type Documentation

enum **_CpaCyKeySslOp**

File: cpa_cy_key.h

SSL Operation Types

Enumeration of the different SSL operations that can be specified in the struct **CpaCyKeyGenSslOpData**. It identifies the label.

Enumerator:

CPA_CY_KEY_SSL_OP_MASTER_SECRET_DERIVE	Derive the master secret
CPA_CY_KEY_SSL_OP_KEY_MATERIAL_DERIVE	Derive the key material
CPA_CY_KEY_SSL_OP_USER_DEFINED	User Defined Operation for custom labels

enum **_CpaCyKeyTlsOp**

File: cpa_cy_key.h

TLS Operation Types

Enumeration of the different TLS operations that can be specified in the CpaCyKeyGenTlsOpData. It identifies the label.

The functions **cpaCyKeyGenTls** and **cpaCyKeyGenTls2** accelerate the TLS PRF, which is defined as part of RFC2246 (TLS v1.0), RFC4346 (TLS v1.1), and RFC5246 (TLS v1.2). One of the inputs to these functions is a label. This enumerated type defines values that correspond to some of the required labels. However, for some of the operations/labels required by these RFCs, no values are specified.

In such cases, a user-defined value must be provided. The client should use the enum value **CPA_CY_KEY_TLS_OP_USER_DEFINED**, and pass the label using the userLabel field of the **CpaCyKeyGenTlsOpData** data structure.

Enumerator:

CPA_CY_KEY_TLS_OP_MASTER_SECRET_DERIVE	Derive the master secret using the TLS PRF. Corresponds to RFC2246/5246 section 8.1, operation "Computing the master secret", label "master secret".
CPA_CY_KEY_TLS_OP_KEY_MATERIAL_DERIVE	Derive the key material using the TLS PRF. Corresponds to RFC2246/5246 section 6.3, operation "Derive the key material", label "key expansion".
CPA_CY_KEY_TLS_OP_CLIENT_FINISHED_DERIVE	Derive the client finished tag using the TLS PRF. Corresponds to RFC2246/5246 section 7.4.9, operation "Client finished", label "client finished".
CPA_CY_KEY_TLS_OP_SERVER_FINISHED_DERIVE	Derive the server finished tag using the TLS PRF. Corresponds to RFC2246/5246 section 7.4.9, operation "Server finished", label "server finished".
CPA_CY_KEY_TLS_OP_USER_DEFINED	User Defined Operation for custom labels.

10.11 Function Documentation

CpaStatus cpaCyKeyGenSsl	(	const **CpaInstanceHandle**	*instanceHandle*,
		const **CpaCyGenFlatBufCbFunc**	*pKeyGenCb*,
		void *	*pCallbackTag*,
		const **CpaCyKeyGenSslOpData** *	*pKeyGenSslOpData*,
		CpaFlatBuffer *	*pGeneratedKeyBuffer*
	)		

File: cpa_cy_key.h

SSL Key Generation Function.

This function is used for SSL key generation. It implements the key generation function defined in section 6.2.2 of the SSL 3.0 specification as described in http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt.

The input seed is taken as a flat buffer and the generated key is returned to caller in a flat destination data buffer.

Context:

When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:

None

Side-Effects:

None

Blocking:

Yes when configured to operate in synchronous mode.

Reentrant:

No

Thread-safe:

Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pKeyGenCb*	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pKeyGenSslOpData*	Structure containing all the data needed to perform the SSL key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.

`[out]`	*pGeneratedKeyBuffer*	Caller MUST allocate a sufficient buffer to hold the key generation output. The data pointer SHOULD be aligned on an 8-byte boundary. The length field passed in represents the size of the buffer in bytes. The value that is returned is the size of the result key in bytes. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized via cpaCyStartInstance function.

Postcondition:

None

See also:

CpaCyKeyGenSslOpData, **CpaCyGenFlatBufCbFunc**

CpaStatus cpaCyKeyGenTls (const **CpaInstanceHandle** *instanceHandle*,
const **CpaCyGenFlatBufCbFunc** *pKeyGenCb*,
void * *pCallbackTag*,
const **CpaCyKeyGenTlsOpData** * *pKeyGenTlsOpData*,
CpaFlatBuffer * *pGeneratedKeyBuffer*
)

File: cpa_cy_key.h

TLS Key Generation Function.

This function is used for TLS key generation. It implements the TLS PRF (Pseudo Random Function) as defined by RFC2246 (TLS v1.0) and RFC4346 (TLS v1.1).

The input seed is taken as a flat buffer and the generated key is returned to caller in a flat destination data buffer.

Context:

When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:

None

Side-Effects:

None

Blocking:

Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pKeyGenCb*	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in]	*pKeyGenTlsOpData*	Structure containing all the data needed to perform the TLS key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pGeneratedKeyBuffer*	Caller MUST allocate a sufficient buffer to hold the key generation output. The data pointer SHOULD be aligned on an 8-byte boundary. The length field passed in represents the size of the buffer in bytes. The value that is returned is the size of the result key in bytes. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

See also:
CpaCyKeyGenTlsOpData, **CpaCyGenFlatBufCbFunc**

```
CpaStatus cpaCyKeyGenTls2 ( const CpaInstanceHandle       instanceHandle,
                            const CpaCyGenFlatBufCbFunc   pKeyGenCb,
                            void *                        pCallbackTag,
                            const CpaCyKeyGenTlsOpData *  pKeyGenTlsOpData,
                            CpaCySymHashAlgorithm         hashAlgorithm,
                            CpaFlatBuffer *               pGeneratedKeyBuffer
                          )
```

File: cpa_cy_key.h

TLS Key Generation Function version 2.

This function is used for TLS key generation. It implements the TLS PRF (Pseudo Random Function) as defined by RFC5246 (TLS v1.2).

The input seed is taken as a flat buffer and the generated key is returned to caller in a flat destination data buffer.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pKeyGenCb*	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pKeyGenTlsOpData*	Structure containing all the data needed to perform the TLS key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[in]`	*hashAlgorithm*	Specifies the hash algorithm to use. According to RFC5246, this should be "SHA-256 or a stronger standard hash function."
`[out]`	*pGeneratedKeyBuffer*	Caller MUST allocate a sufficient buffer to hold the key generation output. The data pointer SHOULD be aligned on an 8-byte boundary. The length field passed in represents the size of the buffer in bytes. The value that is returned is the size of the result key in bytes. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.

CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

See also:
CpaCyKeyGenTlsOpData, **CpaCyGenFlatBufCbFunc**

CpaStatus cpaCyKeyGenMgf	(	const **CpaInstanceHandle**	*instanceHandle*,
		const **CpaCyGenFlatBufCbFunc**	*pKeyGenCb*,
		void *	*pCallbackTag*,
		const **CpaCyKeyGenMgfOpData** *	*pKeyGenMgfOpData*,
		CpaFlatBuffer *	*pGeneratedMaskBuffer*
	)		

File: cpa_cy_key.h

Mask Generation Function.

This function implements the mask generation function MGF1 as defined by PKCS#1 v2.1, and RFC3447. The input seed is taken as a flat buffer and the generated mask is returned to caller in a flat destination data buffer.

Note:
The default hash algorithm used by the MGF is SHA-1. If a different hash algorithm is preferred, then see the "extended" version of this function, **cpaCyKeyGenMgfExt**.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.

`[in]`	*pKeyGenCb*	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pKeyGenMgfOpData*	Structure containing all the data needed to perform the MGF key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pGeneratedMaskBuffer*	Caller MUST allocate a sufficient buffer to hold the generated mask. The data pointer SHOULD be aligned on an 8-byte boundary. The length field passed in represents the size of the buffer in bytes. The value that is returned is the size of the generated mask in bytes. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized via cpaCyStartInstance function.

Postcondition:

None

See also:

CpaCyKeyGenMgfOpData, **CpaCyGenFlatBufCbFunc**

CpaStatus cpaCyKeyGenMgfExt (const **CpaInstanceHandle** *instanceHandle*,
const **CpaCyGenFlatBufCbFunc** *pKeyGenCb*,
void * *pCallbackTag*,
const **CpaCyKeyGenMgfOpDataExt** * *pKeyGenMgfOpDataExt*,
CpaFlatBuffer * *pGeneratedMaskBuffer*
)

File: cpa_cy_key.h

Extended Mask Generation Function.

This function is used for mask generation. It differs from the "base" version of the function (**cpaCyKeyGenMgf**) in that it allows the hash function used by the Mask Generation Function to be specified.

Context:

When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:

None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pKeyGenCb*	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pKeyGenMgfOpDataExt*	Structure containing all the data needed to perform the extended MGF key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pGeneratedMaskBuffer*	Caller MUST allocate a sufficient buffer to hold the generated mask. The data pointer SHOULD be aligned on an 8-byte boundary. The length field passed in represents the size of the buffer in bytes. The value that is returned is the size of the generated mask in bytes. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
This function is only used to generate a mask keys from seed material.

See also:
CpaCyKeyGenMgfOpData, **CpaCyGenFlatBufCbFunc**

CpaStatus CPA_DEPRECATED cpaCyKeyGenQueryStats	(const **CpaInstanceHandle**	*instanceHandle*,
	struct **_CpaCyKeyGenStats** *	*pKeyGenStats*
	)	

File: cpa_cy_key.h

Queries the Key and Mask generation statistics specific to an instance.

Deprecated:
As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyKeyGenQueryStats64()**.

This function will query a specific instance for key and mask generation statistics. The user MUST allocate the CpaCyKeyGenStats structure and pass the reference to that into this function call. This function will write the statistic results into the passed in CpaCyKeyGenStats structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[out]`	*pKeyGenStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:

None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
CpaCyKeyGenStats

CpaStatus cpaCyKeyGenQueryStats64	(	const **CpaInstanceHandle**	*instanceHandle*,
		CpaCyKeyGenStats64 *	*pKeyGenStats*
	)		

File: cpa_cy_key.h

Queries the Key and Mask generation statistics (64-bit version) specific to an instance.

This function will query a specific instance for key and mask generation statistics. The user MUST allocate the CpaCyKeyGenStats64 structure and pass the reference to that into this function call. This function will write the statistic results into the passed in CpaCyKeyGenStats64 structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[out]	*pKeyGenStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
CpaCyKeyGenStats64

11 RSA API

[Cryptographic API]

Collaboration diagram for RSA API:

11.1 Detailed Description

File: cpa_cy_rsa.h

These functions specify the API for Public Key Encryption (Cryptography) RSA operations. The PKCS #1 V2.1 specification is supported, however the support is limited to "two-prime" mode. RSA multi-prime is not supported.

Note:

These functions implement RSA cryptographic primitives. RSA padding schemes are not implemented. For padding schemes that require the mgf function see **Cryptographic Key and Mask Generation API**.

Large numbers are represented on the QuickAssist API as described in the Large Number API (**Cryptographic Large Number API**).

11.2 Data Structures

- struct **_CpaCyRsaPublicKey**
- struct **_CpaCyRsaPrivateKeyRep1**
- struct **_CpaCyRsaPrivateKeyRep2**
- struct **_CpaCyRsaPrivateKey**
- struct **_CpaCyRsaKeyGenOpData**
- struct **_CpaCyRsaEncryptOpData**
- struct **_CpaCyRsaDecryptOpData**
- struct **_CpaCyRsaStats**
- struct **_CpaCyRsaStats64**

11.3 Typedefs

- typedef enum **_CpaCyRsaVersion CpaCyRsaVersion**
- typedef **_CpaCyRsaPublicKey CpaCyRsaPublicKey**
- typedef **_CpaCyRsaPrivateKeyRep1 CpaCyRsaPrivateKeyRep1**
- typedef **_CpaCyRsaPrivateKeyRep2 CpaCyRsaPrivateKeyRep2**
- typedef enum **_CpaCyRsaPrivateKeyRepType CpaCyRsaPrivateKeyRepType**
- typedef **_CpaCyRsaPrivateKey CpaCyRsaPrivateKey**
- typedef **_CpaCyRsaKeyGenOpData CpaCyRsaKeyGenOpData**
- typedef **_CpaCyRsaEncryptOpData CpaCyRsaEncryptOpData**
- typedef **_CpaCyRsaDecryptOpData CpaCyRsaDecryptOpData**
- typedef **_CpaCyRsaStats CPA_DEPRECATED**
- typedef **_CpaCyRsaStats64 CpaCyRsaStats64**
- typedef void(* **CpaCyRsaKeyGenCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pKeyGenOpData, **CpaCyRsaPrivateKey** *pPrivateKey, **CpaCyRsaPublicKey** *pPublicKey)

11.4 Enumerations

- enum **_CpaCyRsaVersion** { **CPA_CY_RSA_VERSION_TWO_PRIME** }
- enum **_CpaCyRsaPrivateKeyRepType** {
  CPA_CY_RSA_PRIVATE_KEY_REP_TYPE_1,
  CPA_CY_RSA_PRIVATE_KEY_REP_TYPE_2
  }

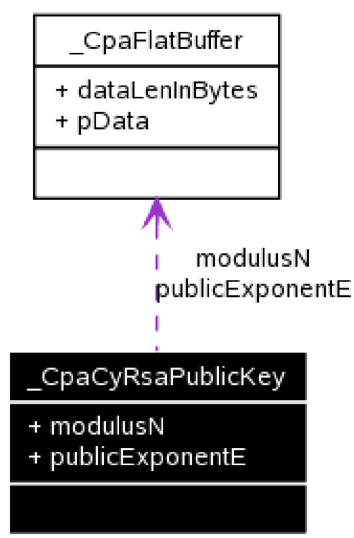
11.5 Functions

- **CpaStatus cpaCyRsaGenKey** (const **CpaInstanceHandle** instanceHandle, const **CpaCyRsaKeyGenCbFunc** pRsaKeyGenCb, void *pCallbackTag, const **CpaCyRsaKeyGenOpData** *pKeyGenOpData, **CpaCyRsaPrivateKey** *pPrivateKey, **CpaCyRsaPublicKey** *pPublicKey)
- **CpaStatus cpaCyRsaEncrypt** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pRsaEncryptCb, void *pCallbackTag, const **CpaCyRsaEncryptOpData** *pEncryptOpData, **CpaFlatBuffer** *pOutputData)
- **CpaStatus cpaCyRsaDecrypt** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pRsaDecryptCb, void *pCallbackTag, const **CpaCyRsaDecryptOpData** *pDecryptOpData, **CpaFlatBuffer** *pOutputData)
- **CpaStatus CPA_DEPRECATED cpaCyRsaQueryStats** (const **CpaInstanceHandle** instanceHandle, struct **_CpaCyRsaStats** *pRsaStats)
- **CpaStatus cpaCyRsaQueryStats64** (const **CpaInstanceHandle** instanceHandle, **CpaCyRsaStats64** *pRsaStats)

11.6 Data Structure Documentation

11.6.1 _CpaCyRsaPublicKey Struct Reference

Collaboration diagram for _CpaCyRsaPublicKey:



11.6.1.1 Detailed Description

File: cpa_cy_rsa.h

RSA Public Key Structure.

11.6.1 _CpaCyRsaPublicKey Struct Reference

This structure contains the two components which comprise the RSA public key as defined in the PKCS #1 V2.1 standard. All values in this structure are required to be in Most Significant Byte first order, e.g. modulusN.pData[0] = MSB.

11.6.1.2 Data Fields

- **CpaFlatBuffer modulusN**
- **CpaFlatBuffer publicExponentE**

11.6.1.3 Field Documentation

CpaFlatBuffer _CpaCyRsaPublicKey::modulusN

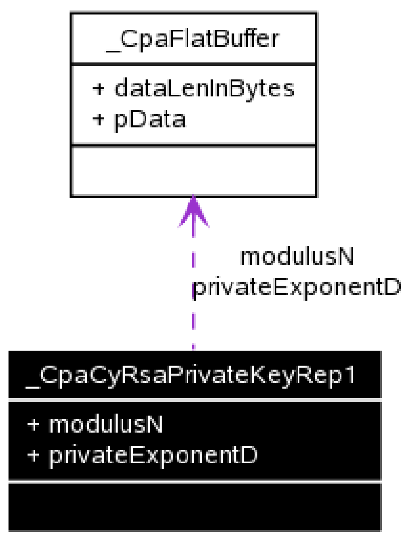
The modulus (n). For key generation operations, the client MUST allocate the memory for this parameter; its value is generated. For encrypt operations this parameter is an input.

CpaFlatBuffer _CpaCyRsaPublicKey::publicExponentE

The public exponent (e). For key generation operations, this field is unused. It is NOT generated by the interface; it is the responsibility of the client to set this to the same value as the corresponding parameter on the CpaCyRsaKeyGenOpData structure before using the key for encryption. For encrypt operations this parameter is an input.

11.6.2 _CpaCyRsaPrivateKeyRep1 Struct Reference

Collaboration diagram for _CpaCyRsaPrivateKeyRep1:



11.6.2.1 Detailed Description

File: cpa_cy_rsa.h

RSA Private Key Structure For Representation 1.

This structure contains the first representation that can be used for describing the RSA private key, represented by the tuple of the modulus (n) and the private exponent (d). All values in this structure are required to be in Most Significant Byte first order, e.g. modulusN.pData[0] = MSB.

11.6.2.2 Data Fields

- **CpaFlatBuffer modulusN**
- **CpaFlatBuffer privateExponentD**

11.6.2.3 Field Documentation

CpaFlatBuffer _CpaCyRsaPrivateKeyRep1::modulusN

The modulus (n). For key generation operations the memory MUST be allocated by the client and the value is generated. For other operations this is an input. Permitted lengths are:

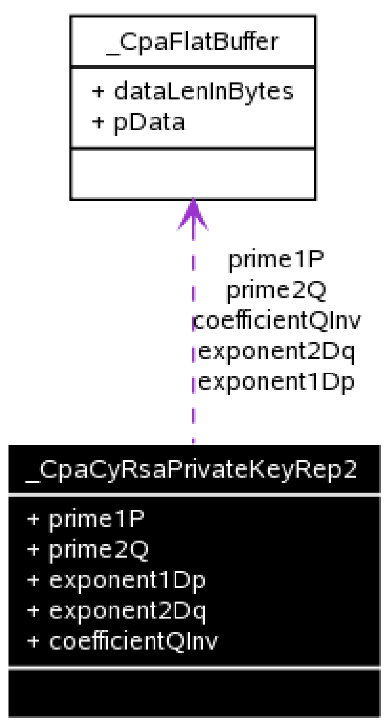
- 512 bits (64 bytes),
- 1024 bits (128 bytes),
- 1536 bits (192 bytes),
- 2048 bits (256 bytes),
- 3072 bits (384 bytes), or
- 4096 bits (512 bytes).

CpaFlatBuffer _CpaCyRsaPrivateKeyRep1::privateExponentD

The private exponent (d). For key generation operations the memory MUST be allocated by the client and the value is generated. For other operations this is an input. NOTE: It is important that the value D is big enough. It is STRONGLY recommended that this value is at least half the length of the modulus N to protect against the Wiener attack.

11.6.3 _CpaCyRsaPrivateKeyRep2 Struct Reference

Collaboration diagram for _CpaCyRsaPrivateKeyRep2:



11.6.3.1 Detailed Description

File: cpa_cy_rsa.h

RSA Private Key Structure For Representation 2.

This structure contains the second representation that can be used for describing the RSA private key. The quintuple of p, q, dP, dQ, and qInv (explained below and in the spec) are required for the second representation. The optional sequence of triplets are not included. All values in this structure are required to be in Most Significant Byte first order, e.g. prime1P.pData[0] = MSB.

11.6.3.2 Data Fields

- **CpaFlatBuffer prime1P**
- **CpaFlatBuffer prime2Q**
- **CpaFlatBuffer exponent1Dp**
- **CpaFlatBuffer exponent2Dq**
- **CpaFlatBuffer coefficientQInv**

11.6.3.3 Field Documentation

CpaFlatBuffer _CpaCyRsaPrivateKeyRep2::prime1P

The first large prime (p). For key generation operations, this field is unused.

CpaFlatBuffer _CpaCyRsaPrivateKeyRep2::prime2Q

The second large prime (q). For key generation operations, this field is unused.

CpaFlatBuffer _CpaCyRsaPrivateKeyRep2::exponent1Dp

The first factor CRT exponent (dP). d mod (p-1).

CpaFlatBuffer _CpaCyRsaPrivateKeyRep2::exponent2Dq

The second factor CRT exponent (dQ). d mod (q-1).

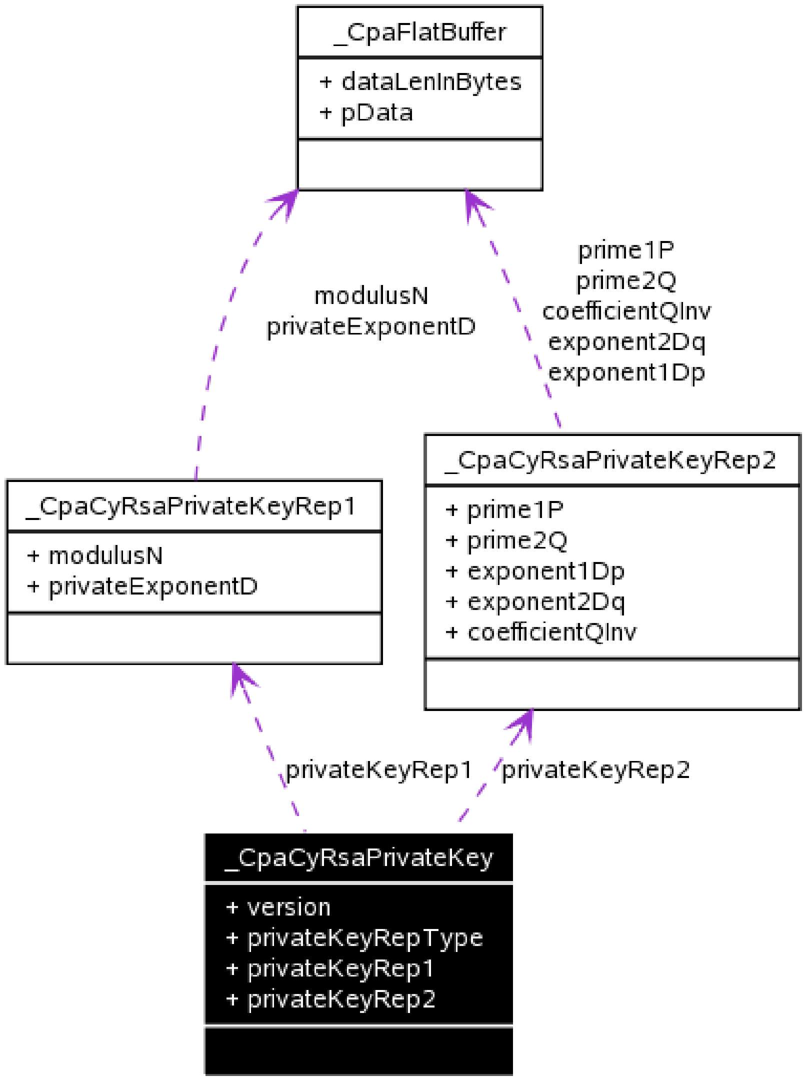
CpaFlatBuffer _CpaCyRsaPrivateKeyRep2::coefficientQInv

The (first) Chinese Remainder Theorem (CRT) coefficient (qInv). (inverse of q) mod p.

11.6.4 _CpaCyRsaPrivateKey Struct Reference

Collaboration diagram for _CpaCyRsaPrivateKey:

11.6.4 _CpaCyRsaPrivateKey Struct Reference



11.6.4.1 Detailed Description

File: cpa_cy_rsa.h

RSA Private Key Structure.

This structure contains the two representations that can be used for describing the RSA private key. The privateKeyRepType will be used to identify which representation is to be used. Typically, using the second representation results in faster decryption operations.

11.6.4.2 Data Fields

- **CpaCyRsaVersion version**
- **CpaCyRsaPrivateKeyRepType privateKeyRepType**
- **CpaCyRsaPrivateKeyRep1 privateKeyRep1**
- **CpaCyRsaPrivateKeyRep2 privateKeyRep2**

11.6.4.3 Field Documentation

CpaCyRsaVersion _CpaCyRsaPrivateKey::version

Indicates the version of the PKCS #1 specification that is supported. Note that this applies to both representations.

CpaCyRsaPrivateKeyRepType _CpaCyRsaPrivateKey::privateKeyRepType

This value is used to identify which of the private key representation types in this structure is relevant. When performing key generation operations for Type 2 representations, memory must also be allocated for the type 1 representations, and values for both will be returned.

CpaCyRsaPrivateKeyRep1 _CpaCyRsaPrivateKey::privateKeyRep1

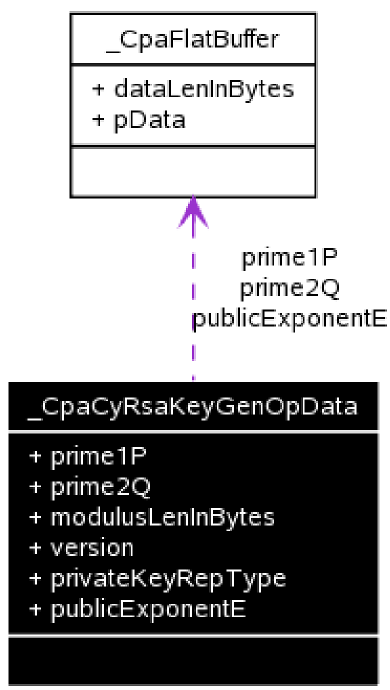
This is the first representation of the RSA private key as defined in the PKCS #1 V2.1 specification. For key generation operations the memory for this structure is allocated by the client and the specific values are generated. For other operations this is an input parameter.

CpaCyRsaPrivateKeyRep2 _CpaCyRsaPrivateKey::privateKeyRep2

This is the second representation of the RSA private key as defined in the PKCS #1 V2.1 specification. For key generation operations the memory for this structure is allocated by the client and the specific values are generated. For other operations this is an input parameter.

11.6.5 _CpaCyRsaKeyGenOpData Struct Reference

Collaboration diagram for _CpaCyRsaKeyGenOpData:



11.6.5.1 Detailed Description

File: cpa_cy_rsa.h

RSA Key Generation Data.

11.6.5 _CpaCyRsaKeyGenOpData Struct Reference

This structure lists the different items that are required in the cpaCyRsaGenKey function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the CpaCyRsaKeyGenCbFunc callback function.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyRsaGenKey function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. prime1P.pData[0] = MSB.

The following limitations on the permutations of the supported bit lengths of p, q and n (written as {p, q, n}) apply:

- {256, 256, 512} or
- {512, 512, 1024} or
- {768, 768, 1536} or
- {1024, 1024, 2048} or
- {1536, 1536, 3072} or
- {2048, 2048, 4096}.

11.6.5.2 Data Fields

- **CpaFlatBuffer prime1P**
- **CpaFlatBuffer prime2Q**
- **Cpa32U modulusLenInBytes**
- **CpaCyRsaVersion version**
- **CpaCyRsaPrivateKeyRepType privateKeyRepType**
- **CpaFlatBuffer publicExponentE**

11.6.5.3 Field Documentation

CpaFlatBuffer _CpaCyRsaKeyGenOpData::prime1P

A large random prime number (p). This MUST be created by the client. Permitted bit lengths are: 256, 512, 768, 1024, 1536 or 2048. Limitations apply - refer to the description above for details.

CpaFlatBuffer _CpaCyRsaKeyGenOpData::prime2Q

A large random prime number (q). This MUST be created by the client. Permitted bit lengths are: 256, 512, 768, 1024, 1536 or 2048. Limitations apply - refer to the description above for details. If the private key representation type is 2, then this pointer will be assigned to the relevant structure member of the representation 2 private key.

Cpa32U _CpaCyRsaKeyGenOpData::modulusLenInBytes

The bit length of the modulus (n). This is the modulus length for both the private and public keys. The length of the modulus N parameter for the private key representation 1 structure and the public key structures will be assigned to this value. References to the strength of RSA actually refer to this bit length. Recommended minimum is 1024 bits. Permitted lengths are:

- 512 bits (64 bytes),
- 1024 bits (128 bytes),
- 1536 bits (192 bytes),
- 2048 bits (256 bytes),
- 3072 bits (384 bytes), or
- 4096 bits (512 bytes). Limitations apply - refer to description above for details.

CpaCyRsaVersion _CpaCyRsaKeyGenOpData::version

Indicates the version of the PKCS #1 specification that is supported. Note that this applies to both representations.

CpaCyRsaPrivateKeyRepType _CpaCyRsaKeyGenOpData::privateKeyRepType

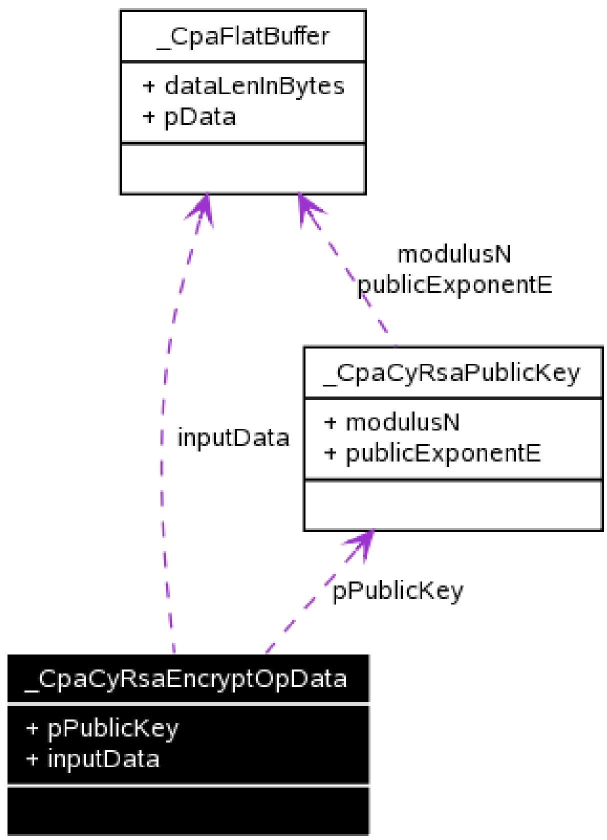
This value is used to identify which of the private key representation types is required to be generated.

CpaFlatBuffer _CpaCyRsaKeyGenOpData::publicExponentE

The public exponent (e).

11.6.6 _CpaCyRsaEncryptOpData Struct Reference

Collaboration diagram for _CpaCyRsaEncryptOpData:



11.6.6.1 Detailed Description

File: cpa_cy_rsa.h

RSA Encryption Primitive Operation Data

This structure lists the different items that are required in the cpaCyRsaEncrypt function. As the RSA encryption primitive and verification primitive operations are mathematically identical this structure may also be used to perform an RSA verification primitive operation. When performing an RSA encryption primitive operation, the input data is the message and the output data is the cipher text. When performing an RSA verification primitive operation, the input data is the signature and the output data is the message. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the CpaCyRsaEncryptCbFunc callback function.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyRsaEncrypt function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. inputData.pData[0] = MSB.

11.6.6.2 Data Fields

- **CpaCyRsaPublicKey * pPublicKey**
- **CpaFlatBuffer inputData**

11.6.6.3 Field Documentation

CpaCyRsaPublicKey* _CpaCyRsaEncryptOpData::pPublicKey

Pointer to the public key.

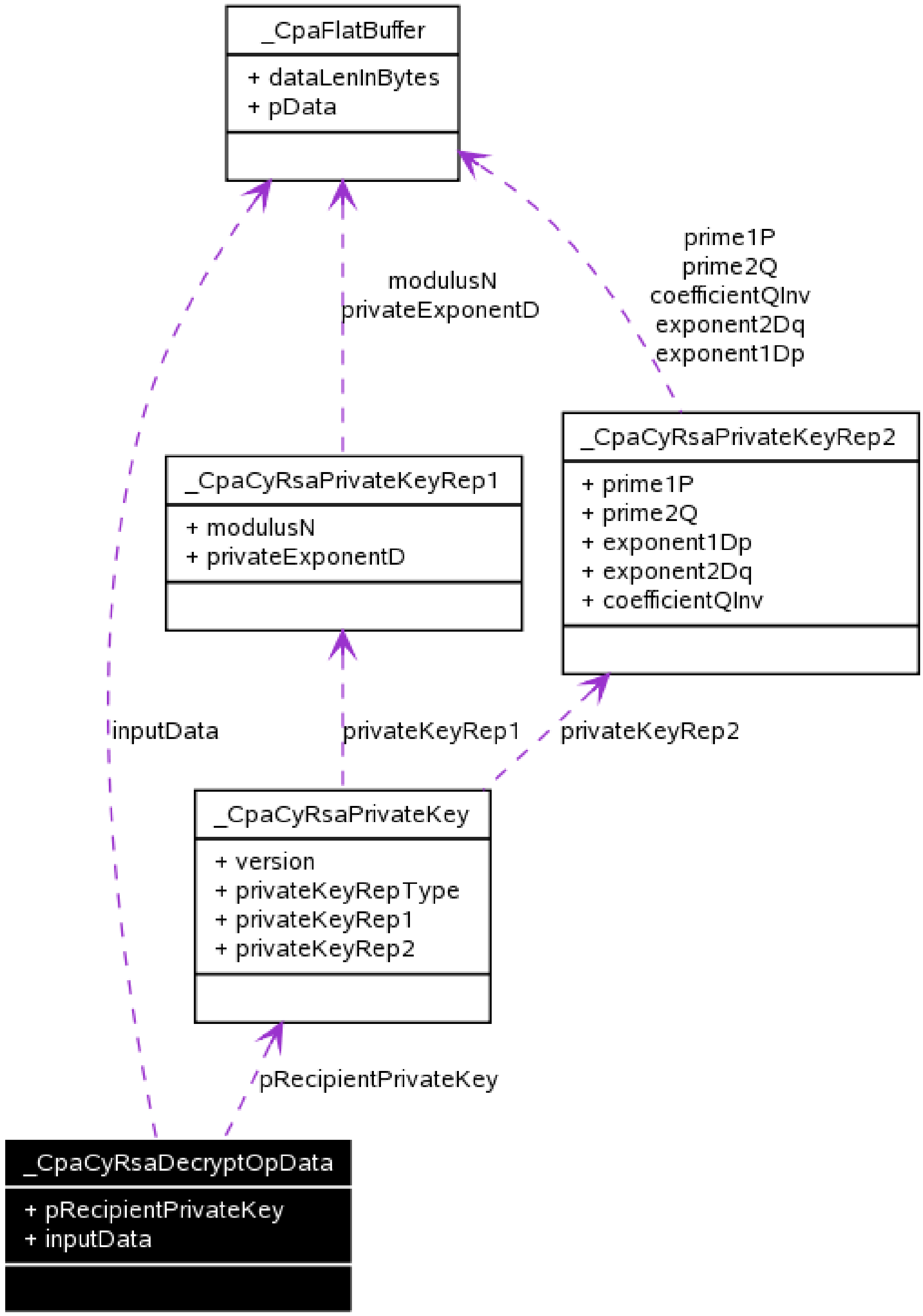
CpaFlatBuffer _CpaCyRsaEncryptOpData::inputData

The input data that the RSA encryption primitive operation is performed on. The data pointed to is an integer that MUST be in big- endian order. The value MUST be between 0 and the modulus n - 1.

11.6.7 _CpaCyRsaDecryptOpData Struct Reference

Collaboration diagram for _CpaCyRsaDecryptOpData:

11.6.7 _CpaCyRsaDecryptOpData Struct Reference



11.6.7.1 Detailed Description

File: cpa_cy_rsa.h

RSA Decryption Primitive Operation Data

This structure lists the different items that are required in the cpaCyRsaDecrypt function. As the RSA decryption primitive and signature primitive operations are mathematically identical this structure may also be used to perform an RSA signature primitive operation. When performing an RSA decryption primitive operation, the input data is the cipher text and the output data is the message text. When performing an RSA signature primitive operation, the input data is the message and the output data is the signature. The client

MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to he function. Ownership of the memory returns to the client when this structure is returned in the CpaCyRsaDecryptCbFunc callback function.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyRsaDecrypt function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. inputData.pData[0] = MSB.

11.6.7.2 Data Fields

- **CpaCyRsaPrivateKey * pRecipientPrivateKey**
- **CpaFlatBuffer inputData**

11.6.7.3 Field Documentation

CpaCyRsaPrivateKey* _CpaCyRsaDecryptOpData::pRecipientPrivateKey

Pointer to the recipient's RSA private key.

CpaFlatBuffer _CpaCyRsaDecryptOpData::inputData

The input data that the RSA decryption primitive operation is performed on. The data pointed to is an integer that MUST be in big- endian order. The value MUST be between 0 and the modulus n - 1.

11.6.8 _CpaCyRsaStats Struct Reference

11.6.8.1 Detailed Description

File: cpa_cy_rsa.h

RSA Statistics.

Deprecated:

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyRsaStats64**.

This structure contains statistics on the RSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

11.6.8.2 Data Fields

- **Cpa32U numRsaKeyGenRequests**
- **Cpa32U numRsaKeyGenRequestErrors**
- **Cpa32U numRsaKeyGenCompleted**
- **Cpa32U numRsaKeyGenCompletedErrors**
- **Cpa32U numRsaEncryptRequests**
- **Cpa32U numRsaEncryptRequestErrors**
- **Cpa32U numRsaEncryptCompleted**
- **Cpa32U numRsaEncryptCompletedErrors**
- **Cpa32U numRsaDecryptRequests**
- **Cpa32U numRsaDecryptRequestErrors**
- **Cpa32U numRsaDecryptCompleted**
- **Cpa32U numRsaDecryptCompletedErrors**

11.6.8.3 Field Documentation

Cpa32U _CpaCyRsaStats::numRsaKeyGenRequests

Total number of successful RSA key generation requests.

Cpa32U _CpaCyRsaStats::numRsaKeyGenRequestErrors

Total number of RSA key generation requests that had an error and could not be processed.

Cpa32U _CpaCyRsaStats::numRsaKeyGenCompleted

Total number of RSA key generation operations that completed successfully.

Cpa32U _CpaCyRsaStats::numRsaKeyGenCompletedErrors

Total number of RSA key generation operations that could not be completed successfully due to errors.

Cpa32U _CpaCyRsaStats::numRsaEncryptRequests

Total number of successful RSA encrypt operation requests.

Cpa32U _CpaCyRsaStats::numRsaEncryptRequestErrors

Total number of RSA encrypt requests that had an error and could not be processed.

Cpa32U _CpaCyRsaStats::numRsaEncryptCompleted

Total number of RSA encrypt operations that completed successfully.

Cpa32U _CpaCyRsaStats::numRsaEncryptCompletedErrors

Total number of RSA encrypt operations that could not be completed successfully due to errors.

Cpa32U _CpaCyRsaStats::numRsaDecryptRequests

Total number of successful RSA decrypt operation requests.

Cpa32U _CpaCyRsaStats::numRsaDecryptRequestErrors

Total number of RSA decrypt requests that had an error and could not be processed.

Cpa32U _CpaCyRsaStats::numRsaDecryptCompleted

Total number of RSA decrypt operations that completed successfully.

Cpa32U _CpaCyRsaStats::numRsaDecryptCompletedErrors

Total number of RSA decrypt operations that could not be completed successfully due to errors.

11.6.9 _CpaCyRsaStats64 Struct Reference

11.6.9.1 Detailed Description

File: cpa_cy_rsa.h

RSA Statistics (64-bit version).

This structure contains 64-bit version of the statistics on the RSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

11.6.9.2 Data Fields

- **Cpa64U numRsaKeyGenRequests**
- **Cpa64U numRsaKeyGenRequestErrors**
- **Cpa64U numRsaKeyGenCompleted**
- **Cpa64U numRsaKeyGenCompletedErrors**
- **Cpa64U numRsaEncryptRequests**
- **Cpa64U numRsaEncryptRequestErrors**
- **Cpa64U numRsaEncryptCompleted**
- **Cpa64U numRsaEncryptCompletedErrors**
- **Cpa64U numRsaDecryptRequests**
- **Cpa64U numRsaDecryptRequestErrors**
- **Cpa64U numRsaDecryptCompleted**
- **Cpa64U numRsaDecryptCompletedErrors**

11.6.9.3 Field Documentation

Cpa64U _CpaCyRsaStats64::numRsaKeyGenRequests

Total number of successful RSA key generation requests.

Cpa64U _CpaCyRsaStats64::numRsaKeyGenRequestErrors

Total number of RSA key generation requests that had an error and could not be processed.

Cpa64U _CpaCyRsaStats64::numRsaKeyGenCompleted

Total number of RSA key generation operations that completed successfully.

Cpa64U _CpaCyRsaStats64::numRsaKeyGenCompletedErrors

Total number of RSA key generation operations that could not be completed successfully due to errors.

Cpa64U _CpaCyRsaStats64::numRsaEncryptRequests

Total number of successful RSA encrypt operation requests.

Cpa64U _CpaCyRsaStats64::numRsaEncryptRequestErrors

Total number of RSA encrypt requests that had an error and could not be processed.

Cpa64U _CpaCyRsaStats64::numRsaEncryptCompleted

Total number of RSA encrypt operations that completed successfully.

Cpa64U _CpaCyRsaStats64::numRsaEncryptCompletedErrors

Total number of RSA encrypt operations that could not be completed successfully due to errors.

Cpa64U _CpaCyRsaStats64::numRsaDecryptRequests

Total number of successful RSA decrypt operation requests.

Cpa64U _CpaCyRsaStats64::numRsaDecryptRequestErrors

Total number of RSA decrypt requests that had an error and could not be processed.

Cpa64U _CpaCyRsaStats64::numRsaDecryptCompleted

Total number of RSA decrypt operations that completed successfully.

Cpa64U _CpaCyRsaStats64::numRsaDecryptCompletedErrors

Total number of RSA decrypt operations that could not be completed successfully due to errors.

11.7 Typedef Documentation

typedef enum **_CpaCyRsaVersion CpaCyRsaVersion**

File: cpa_cy_rsa.h

RSA Version.

This enumeration lists the version identifier for the PKCS #1 V2.1 standard.

Note:
Multi-prime (more than two primes) is not supported.

typedef struct **_CpaCyRsaPublicKey CpaCyRsaPublicKey**

File: cpa_cy_rsa.h

RSA Public Key Structure.

This structure contains the two components which comprise the RSA public key as defined in the PKCS #1 V2.1 standard. All values in this structure are required to be in Most Significant Byte first order, e.g. modulusN.pData[0] = MSB.

typedef struct **_CpaCyRsaPrivateKeyRep1 CpaCyRsaPrivateKeyRep1**

File: cpa_cy_rsa.h

RSA Private Key Structure For Representation 1.

This structure contains the first representation that can be used for describing the RSA private key, represented by the tuple of the modulus (n) and the private exponent (d). All values in this structure are required to be in Most Significant Byte first order, e.g. modulusN.pData[0] = MSB.

typedef struct **_CpaCyRsaPrivateKeyRep2 CpaCyRsaPrivateKeyRep2**

File: cpa_cy_rsa.h

RSA Private Key Structure For Representation 2.

This structure contains the second representation that can be used for describing the RSA private key. The quintuple of p, q, dP, dQ, and qInv (explained below and in the spec) are required for the second representation. The optional sequence of triplets are not included. All values in this structure are required to be in Most Significant Byte first order, e.g. prime1P.pData[0] = MSB.

typedef enum **_CpaCyRsaPrivateKeyRepType CpaCyRsaPrivateKeyRepType**

File: cpa_cy_rsa.h

RSA private key representation type.

This enumeration lists which PKCS V2.1 representation of the private key is being used.

typedef struct **_CpaCyRsaPrivateKey CpaCyRsaPrivateKey**

File: cpa_cy_rsa.h

RSA Private Key Structure.

This structure contains the two representations that can be used for describing the RSA private key. The privateKeyRepType will be used to identify which representation is to be used. Typically, using the second representation results in faster decryption operations.

typedef struct **_CpaCyRsaKeyGenOpData CpaCyRsaKeyGenOpData**

File: cpa_cy_rsa.h

RSA Key Generation Data.

This structure lists the different items that are required in the cpaCyRsaGenKey function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the CpaCyRsaKeyGenCbFunc callback function.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyRsaGenKey function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. prime1P.pData[0] = MSB.

The following limitations on the permutations of the supported bit lengths of p, q and n (written as {p, q, n}) apply:

- {256, 256, 512} or
- {512, 512, 1024} or
- {768, 768, 1536} or
- {1024, 1024, 2048} or
- {1536, 1536, 3072} or
- {2048, 2048, 4096}.

typedef struct **_CpaCyRsaEncryptOpData CpaCyRsaEncryptOpData**

File: cpa_cy_rsa.h

RSA Encryption Primitive Operation Data

This structure lists the different items that are required in the cpaCyRsaEncrypt function. As the RSA encryption primitive and verification primitive operations are mathematically identical this structure may also be used to perform an RSA verification primitive operation. When performing an RSA encryption primitive operation, the input data is the message and the output data is the cipher text. When performing an RSA verification primitive operation, the input data is the signature and the output data is the message. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the CpaCyRsaEncryptCbFunc callback function.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyRsaEncrypt function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. inputData.pData[0] = MSB.

typedef struct **_CpaCyRsaDecryptOpData CpaCyRsaDecryptOpData**

File: cpa_cy_rsa.h

RSA Decryption Primitive Operation Data

This structure lists the different items that are required in the cpaCyRsaDecrypt function. As the RSA decryption primitive and signature primitive operations are mathematically identical this structure may also be used to perform an RSA signature primitive operation. When performing an RSA decryption primitive operation, the input data is the cipher text and the output data is the message text. When performing an RSA signature primitive operation, the input data is the message and the output data is the signature. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to he function. Ownership of the memory returns to the client when this structure is returned in the CpaCyRsaDecryptCbFunc callback function.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyRsaDecrypt function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. inputData.pData[0] = MSB.

typedef struct **_CpaCyRsaStats CPA_DEPRECATED**

File: cpa_cy_rsa.h

RSA Statistics.

Deprecated:

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyRsaStats64**.

This structure contains statistics on the RSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef struct **_CpaCyRsaStats64 CpaCyRsaStats64**

File: cpa_cy_rsa.h

RSA Statistics (64-bit version).

This structure contains 64-bit version of the statistics on the RSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef void(* **CpaCyRsaKeyGenCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pKeyGenOpData, **CpaCyRsaPrivateKey** *pPrivateKey, **CpaCyRsaPublicKey** *pPublicKey)

File: cpa_cy_rsa.h

Definition of the RSA key generation callback function.

This is the prototype for the RSA key generation callback function. The callback function pointer is passed in as a parameter to the cpaCyRsaGenKey function. It will be invoked once the request has completed.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*pCallbackTag*	Opaque value provided by user while making individual function calls.
[in]	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
[in]	*pKeyGenOpData*	Structure with output params for callback.
[in]	*pPrivateKey*	Structure which contains pointers to the memory into which the generated private key will be written.
[in]	*pPublicKey*	Structure which contains pointers to the memory into which the generated public key will be written. The pointer to the public exponent (e) that is returned in this structure is equal to the input public exponent.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
CpaCyRsaPrivateKey, **CpaCyRsaPublicKey**, **cpaCyRsaGenKey()**

11.8 Enumeration Type Documentation

enum **_CpaCyRsaVersion**

File: cpa_cy_rsa.h

RSA Version.

This enumeration lists the version identifier for the PKCS #1 V2.1 standard.

Note:
Multi-prime (more than two primes) is not supported.

Enumerator:

CPA_CY_RSA_VERSION_TWO_PRIME	The version supported is "two-prime".

enum **_CpaCyRsaPrivateKeyRepType**

File: cpa_cy_rsa.h

RSA private key representation type.

This enumeration lists which PKCS V2.1 representation of the private key is being used.

Enumerator:

CPA_CY_RSA_PRIVATE_KEY_REP_TYPE_1	The first representation of the RSA private key.
CPA_CY_RSA_PRIVATE_KEY_REP_TYPE_2	The second representation of the RSA private key.

11.9 Function Documentation

CpaStatus cpaCyRsaGenKey (

const **CpaInstanceHandle**	*instanceHandle*,
const **CpaCyRsaKeyGenCbFunc**	*pRsaKeyGenCb*,
void *	*pCallbackTag*,
const **CpaCyRsaKeyGenOpData** *	*pKeyGenOpData*,
CpaCyRsaPrivateKey *	*pPrivateKey*,
CpaCyRsaPublicKey *	*pPublicKey*

)

File: cpa_cy_rsa.h

Generate RSA keys.

This function will generate private and public keys for RSA as specified in the PKCS #1 V2.1 standard. Both representation types of the private key may be generated.

Context:

When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:

None

Side-Effects:

None

Blocking:

Yes when configured to operate in synchronous mode.

Reentrant:

No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pRsaKeyGenCb*	Pointer to the callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pKeyGenOpData*	Structure containing all the data needed to perform the RSA key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pPrivateKey*	Structure which contains pointers to the memory into which the generated private key will be written. The client MUST allocate memory for this structure, and for the pointers within it, recursively; on return, these will be populated.
`[out]`	*pPublicKey*	Structure which contains pointers to the memory into which the generated public key will be written. The memory for this structure and for the modulusN parameter MUST be allocated by the client, and will be populated on return from the call. The field publicExponentE is not modified or touched in any way; it is the responsibility of the client to set this to the same value as the corresponding parameter on the CpaCyRsaKeyGenOpData structure before using the key for encryption.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pRsaKeyGenCb is non-NULL, an asynchronous callback of type is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyRsaKeyGenOpData, **CpaCyRsaKeyGenCbFunc**, **cpaCyRsaEncrypt()**, **cpaCyRsaDecrypt()**

CpaStatus cpaCyRsaEncrypt	(	const **CpaInstanceHandle**	*instanceHandle*,
		const **CpaCyGenFlatBufCbFunc**	*pRsaEncryptCb*,
		void *	*pCallbackTag*,
		const **CpaCyRsaEncryptOpData** *	*pEncryptOpData*,
		CpaFlatBuffer *	*pOutputData*
	)		

File: cpa_cy_rsa.h

Perform the RSA encrypt (or verify) primitive operation on the input data.

This function will perform an RSA encryption primitive operation on the input data using the specified RSA public key. As the RSA encryption primitive and verification primitive operations are mathematically identical this function may also be used to perform an RSA verification primitive operation.

Context:
: When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
: None

Side-Effects:
: None

Blocking:
: Yes when configured to operate in synchronous mode.

Reentrant:
: No

Thread-safe:
: Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pRsaEncryptCb*	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pEncryptOpData*	Structure containing all the data needed to perform the RSA encryption operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pOutputData*	Pointer to structure into which the result of the RSA encryption primitive is written. The client MUST allocate this memory. The data pointed to is an integer in big-endian order. The value will be between 0 and the modulus n - 1. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pRsaEncryptCb is non-NULL an asynchronous callback of type is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyGenFlatBufCbFunc CpaCyRsaEncryptOpData cpaCyRsaGenKey() cpaCyRsaDecrypt()

CpaStatus cpaCyRsaDecrypt (const **CpaInstanceHandle** *instanceHandle*,
const **CpaCyGenFlatBufCbFunc** *pRsaDecryptCb*,
void * *pCallbackTag*,
const **CpaCyRsaDecryptOpData** * *pDecryptOpData*,
CpaFlatBuffer * *pOutputData*
)

File: cpa_cy_rsa.h

Perform the RSA decrypt (or sign) primitive operation on the input data.

This function will perform an RSA decryption primitive operation on the input data using the specified RSA private key. As the RSA decryption primitive and signing primitive operations are mathematically identical this function may also be used to perform an RSA signing primitive operation.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pRsaDecryptCb*	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pDecryptOpData*	Structure containing all the data needed to perform the RSA decrypt operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pOutputData*	Pointer to structure into which the result of the RSA decryption primitive is written. The client MUST allocate this memory. The data pointed to is an integer in big-endian order. The value will be between 0 and the modulus n - 1. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pRsaDecryptCb is non-NULL an asynchronous callback is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyRsaDecryptOpData, **CpaCyGenFlatBufCbFunc**, **cpaCyRsaGenKey()**, **cpaCyRsaEncrypt()**

CpaStatus CPA_DEPRECATED cpaCyRsaQueryStats (const **CpaInstanceHandle** *instanceHandle*, struct **_CpaCyRsaStats** * *pRsaStats*)

File: cpa_cy_rsa.h

Query statistics for a specific RSA instance.

Deprecated:
As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyRsaQueryStats64()**.

This function will query a specific instance for RSA statistics. The user MUST allocate the CpaCyRsaStats structure and pass the reference to that into this function call. This function will write the statistic results into the passed in CpaCyRsaStats structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[out]`	*pRsaStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
CpaCyRsaStats

CpaStatus cpaCyRsaQueryStats64 (	const **CpaInstanceHandle**	*instanceHandle*,
	CpaCyRsaStats64 *	*pRsaStats*
)		

File: cpa_cy_rsa.h

Query statistics (64-bit version) for a specific RSA instance.

This function will query a specific instance for RSA statistics. The user MUST allocate the CpaCyRsaStats64 structure and pass the reference to that into this function call. This function will write the statistic results into the passed in CpaCyRsaStats64 structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
: This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
: None

Side-Effects:
: None

Blocking:
: This function is synchronous and blocking.

Reentrant:
: No

Thread-safe:
: Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[out]`	*pRsaStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
: Component has been initialized.

Postcondition:
: None

Note:
: This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
: **CpaCyRsaStats64**

12 Diffie-Hellman (DH) API

[Cryptographic API]

Collaboration diagram for Diffie-Hellman (DH) API:

12.1 Detailed Description

File: cpa_cy_dh.h

These functions specify the API for Public Key Encryption (Cryptography) operations for use with Diffie-Hellman algorithm.

Note:

Large numbers are represented on the QuickAssist API as described in the Large Number API (**Cryptographic Large Number API**).

12.2 Data Structures

- struct **_CpaCyDhPhase1KeyGenOpData**
- struct **_CpaCyDhPhase2SecretKeyGenOpData**
- struct **_CpaCyDhStats**
- struct **_CpaCyDhStats64**

12.3 Typedefs

- typedef **_CpaCyDhPhase1KeyGenOpData CpaCyDhPhase1KeyGenOpData**
- typedef **_CpaCyDhPhase2SecretKeyGenOpData CpaCyDhPhase2SecretKeyGenOpData**
- typedef **_CpaCyDhStats CPA_DEPRECATED**
- typedef **_CpaCyDhStats64 CpaCyDhStats64**

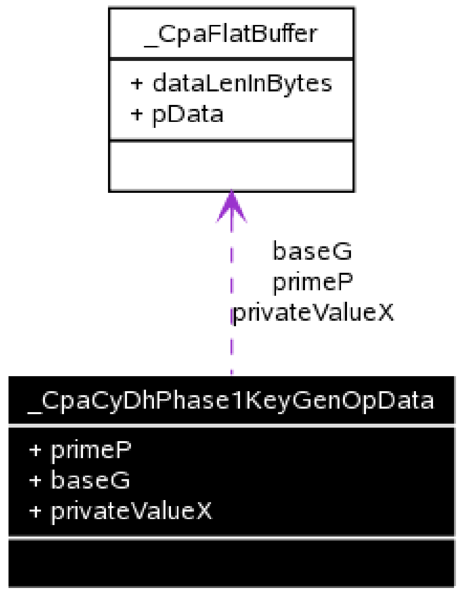
12.4 Functions

- **CpaStatus cpaCyDhKeyGenPhase1** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pDhPhase1Cb, void *pCallbackTag, const **CpaCyDhPhase1KeyGenOpData** *pPhase1KeyGenData, **CpaFlatBuffer** *pLocalOctetStringPV)
- **CpaStatus cpaCyDhKeyGenPhase2Secret** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pDhPhase2Cb, void *pCallbackTag, const **CpaCyDhPhase2SecretKeyGenOpData** *pPhase2SecretKeyGenData, **CpaFlatBuffer** *pOctetStringSecretKey)
- **CpaStatus CPA_DEPRECATED cpaCyDhQueryStats** (const **CpaInstanceHandle** instanceHandle, struct **_CpaCyDhStats** *pDhStats)
- **CpaStatus cpaCyDhQueryStats64** (const **CpaInstanceHandle** instanceHandle, **CpaCyDhStats64** *pDhStats)

12.5 Data Structure Documentation

12.5.1 _CpaCyDhPhase1KeyGenOpData Struct Reference

Collaboration diagram for _CpaCyDhPhase1KeyGenOpData:



12.5.1.1 Detailed Description

File: cpa_cy_dh.h

Diffie-Hellman Phase 1 Key Generation Data.

This structure lists the different items that are required in the cpaCyDhKeyGenPhase1 function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the CpaCyDhPhase1KeyGenOpData structure.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDhKeyGenPhase1 function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. primeP.pData[0] = MSB.

12.5.1.2 Data Fields

- **CpaFlatBuffer primeP**
- **CpaFlatBuffer baseG**
- **CpaFlatBuffer privateValueX**

12.5.1.3 Field Documentation

CpaFlatBuffer _CpaCyDhPhase1KeyGenOpData::primeP

Flat buffer containing a pointer to the random odd prime number (p). The bit-length of this number may be one of 768, 1024, 1536, 2048, 3072 or 4096.

CpaFlatBuffer _CpaCyDhPhase1KeyGenOpData::baseG

Flat buffer containing a pointer to base (g). This MUST comply with the following: 0 < g < p.

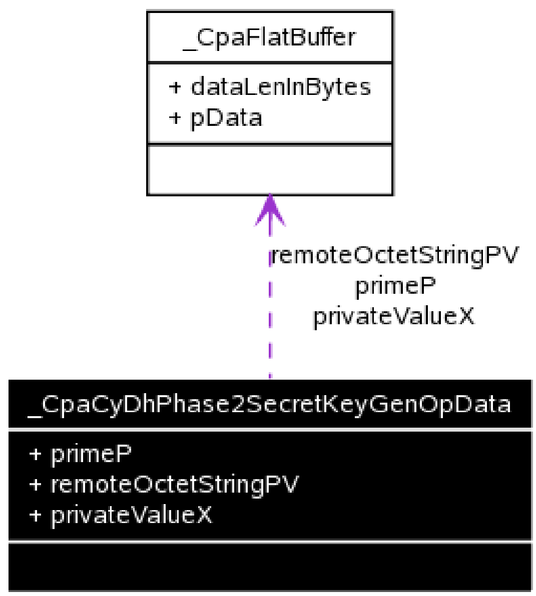
CpaFlatBuffer _CpaCyDhPhase1KeyGenOpData::privateValueX

Flat buffer containing a pointer to the private value (x). This is a random value which MUST satisfy the following condition: 0 < PrivateValueX < (PrimeP - 1)

Refer to PKCS #3: Diffie-Hellman Key-Agreement Standard for details. The client creating this data MUST ensure the compliance of this value with the standard. Note: This value is also needed to complete local phase 2 Diffie-Hellman operation.

12.5.2 _CpaCyDhPhase2SecretKeyGenOpData Struct Reference

Collaboration diagram for _CpaCyDhPhase2SecretKeyGenOpData:



12.5.2.1 Detailed Description

File: cpa_cy_dh.h

Diffie-Hellman Phase 2 Secret Key Generation Data.

This structure lists the different items that required in the cpaCyDhKeyGenPhase2Secret function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDhKeyGenPhase2Secret function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. primeP.pData[0] = MSB.

12.5.2.2 Data Fields

- **CpaFlatBuffer primeP**
- **CpaFlatBuffer remoteOctetStringPV**
- **CpaFlatBuffer privateValueX**

12.5.2.3 Field Documentation

CpaFlatBuffer _CpaCyDhPhase2SecretKeyGenOpData::primeP

Flat buffer containing a pointer to the random odd prime number (p). The bit-length of this number may be one of 768, 1024, 1536, 2048, 3072 or 4096. This SHOULD be same prime number as was used in the phase 1 key generation operation.

CpaFlatBuffer _CpaCyDhPhase2SecretKeyGenOpData::remoteOctetStringPV

Flat buffer containing a pointer to the remote entity octet string Public Value (PV).

CpaFlatBuffer _CpaCyDhPhase2SecretKeyGenOpData::privateValueX

Flat buffer containing a pointer to the private value (x). This value may have been used in a call to the cpaCyDhKeyGenPhase1 function. This is a random value which MUST satisfy the following condition: 0 < privateValueX < (primeP - 1).

12.5.3 _CpaCyDhStats Struct Reference

12.5.3.1 Detailed Description

File: cpa_cy_dh.h

Diffie-Hellman Statistics.

Deprecated:

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyDhStats64**.

This structure contains statistics on the Diffie-Hellman operations. Statistics are set to zero when the component is initialized, and are collected per instance.

12.5.3.2 Data Fields

- **Cpa32U numDhPhase1KeyGenRequests**
- **Cpa32U numDhPhase1KeyGenRequestErrors**
- **Cpa32U numDhPhase1KeyGenCompleted**
- **Cpa32U numDhPhase1KeyGenCompletedErrors**
- **Cpa32U numDhPhase2KeyGenRequests**
- **Cpa32U numDhPhase2KeyGenRequestErrors**
- **Cpa32U numDhPhase2KeyGenCompleted**
- **Cpa32U numDhPhase2KeyGenCompletedErrors**

12.5.3.3 Field Documentation

Cpa32U _CpaCyDhStats::numDhPhase1KeyGenRequests

Total number of successful Diffie-Hellman phase 1 key generation requests.

Cpa32U _CpaCyDhStats::numDhPhase1KeyGenRequestErrors

Total number of Diffie-Hellman phase 1 key generation requests that had an error and could not be processed.

Cpa32U _CpaCyDhStats::numDhPhase1KeyGenCompleted

Total number of Diffie-Hellman phase 1 key generation operations that completed successfully.

Cpa32U _CpaCyDhStats::numDhPhase1KeyGenCompletedErrors

Total number of Diffie-Hellman phase 1 key generation operations that could not be completed successfully due to errors.

Cpa32U _CpaCyDhStats::numDhPhase2KeyGenRequests

Total number of successful Diffie-Hellman phase 2 key generation requests.

Cpa32U _CpaCyDhStats::numDhPhase2KeyGenRequestErrors

Total number of Diffie-Hellman phase 2 key generation requests that had an error and could not be processed.

Cpa32U _CpaCyDhStats::numDhPhase2KeyGenCompleted

Total number of Diffie-Hellman phase 2 key generation operations that completed successfully.

Cpa32U _CpaCyDhStats::numDhPhase2KeyGenCompletedErrors

Total number of Diffie-Hellman phase 2 key generation operations that could not be completed successfully due to errors.

12.5.4 _CpaCyDhStats64 Struct Reference

12.5.4.1 Detailed Description

File: cpa_cy_dh.h

Diffie-Hellman Statistics (64-bit version).

This structure contains the 64-bit version of the statistics on the Diffie-Hellman operations. Statistics are set to zero when the component is initialized, and are collected per instance.

12.5.4.2 Data Fields

- **Cpa64U numDhPhase1KeyGenRequests**
- **Cpa64U numDhPhase1KeyGenRequestErrors**
- **Cpa64U numDhPhase1KeyGenCompleted**
- **Cpa64U numDhPhase1KeyGenCompletedErrors**
- **Cpa64U numDhPhase2KeyGenRequests**
- **Cpa64U numDhPhase2KeyGenRequestErrors**
- **Cpa64U numDhPhase2KeyGenCompleted**
- **Cpa64U numDhPhase2KeyGenCompletedErrors**

12.5.4.3 Field Documentation

Cpa64U _CpaCyDhStats64::numDhPhase1KeyGenRequests

Total number of successful Diffie-Hellman phase 1 key generation requests.

Cpa64U _CpaCyDhStats64::numDhPhase1KeyGenRequestErrors

Total number of Diffie-Hellman phase 1 key generation requests that had an error and could not be processed.

Cpa64U _CpaCyDhStats64::numDhPhase1KeyGenCompleted

Total number of Diffie-Hellman phase 1 key generation operations that completed successfully.

Cpa64U _CpaCyDhStats64::numDhPhase1KeyGenCompletedErrors

Total number of Diffie-Hellman phase 1 key generation operations that could not be completed successfully due to errors.

Cpa64U _CpaCyDhStats64::numDhPhase2KeyGenRequests

Total number of successful Diffie-Hellman phase 2 key generation requests.

Cpa64U _CpaCyDhStats64::numDhPhase2KeyGenRequestErrors

Total number of Diffie-Hellman phase 2 key generation requests that had an error and could not be processed.

Cpa64U _CpaCyDhStats64::numDhPhase2KeyGenCompleted

Total number of Diffie-Hellman phase 2 key generation operations that completed successfully.

Cpa64U _CpaCyDhStats64::numDhPhase2KeyGenCompletedErrors

Total number of Diffie-Hellman phase 2 key generation operations that could not be completed successfully due to errors.

12.6 Typedef Documentation

typedef struct **_CpaCyDhPhase1KeyGenOpData CpaCyDhPhase1KeyGenOpData**

File: cpa_cy_dh.h

Diffie-Hellman Phase 1 Key Generation Data.

This structure lists the different items that are required in the cpaCyDhKeyGenPhase1 function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the CpaCyDhPhase1KeyGenOpData structure.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDhKeyGenPhase1 function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. primeP.pData[0] = MSB.

typedef struct **_CpaCyDhPhase2SecretKeyGenOpData CpaCyDhPhase2SecretKeyGenOpData**

File: cpa_cy_dh.h

Diffie-Hellman Phase 2 Secret Key Generation Data.

This structure lists the different items that required in the cpaCyDhKeyGenPhase2Secret function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDhKeyGenPhase2Secret function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. primeP.pData[0] = MSB.

typedef struct **_CpaCyDhStats CPA_DEPRECATED**

File: cpa_cy_dh.h

Diffie-Hellman Statistics.

Deprecated:
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyDhStats64**.

This structure contains statistics on the Diffie-Hellman operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef struct **_CpaCyDhStats64 CpaCyDhStats64**

File: cpa_cy_dh.h

Diffie-Hellman Statistics (64-bit version).

This structure contains the 64-bit version of the statistics on the Diffie-Hellman operations. Statistics are set to zero when the component is initialized, and are collected per instance.

12.7 Function Documentation

CpaStatus cpaCyDhKeyGenPhase1 (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyGenFlatBufCbFunc**	*pDhPhase1Cb*,
	void *	*pCallbackTag*,
	const **CpaCyDhPhase1KeyGenOpData** *	*pPhase1KeyGenData*,
	CpaFlatBuffer *	*pLocalOctetStringPV*
	)	

File: cpa_cy_dh.h

Function to implement Diffie-Hellman phase 1 operations.

This function may be used to implement the Diffie-Hellman phase 1 operations as defined in the PKCS #3 standard. It may be used to generate the the (local) octet string public value (PV) key. The prime number sizes specified in RFC 2409, 4306, and part of RFC 3526 are supported (bit sizes 6144 and 8192 from RFC 3536 are not supported).

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pDhPhase1Cb*	Pointer to a callback function to be invoked when the operation is complete. If the pointer is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback
[in]	*pPhase1KeyGenData*	Structure containing all the data needed to perform the DH Phase 1 key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pLocalOctetStringPV*	Pointer to memory allocated by the client into which the (local) octet string Public Value (PV) will be written. This value needs to be sent to the remote entity with which Diffie-Hellman is negotiating. The size of this buffer in bytes (as represented by the dataLenInBytes field) MUST be at least big enough to store the public value, which may have a bit length up to that of pPrimeP. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pDhPhase1Cb is non-NULL an asynchronous callback of type CpaCyGenFlatBufCbFunc is generated in response to this function call. Any errors generated during processing are reported in the structure returned in the callback.

See also:

CpaCyGenFlatBufCbFunc, **CpaCyDhPhase1KeyGenOpData**

CpaStatus cpaCyDhKeyGenPhase2Secret	(	const **CpaInstanceHandle**	*instanceHandle*,
		const **CpaCyGenFlatBufCbFunc**	*pDhPhase2Cb*,
		void *	*pCallbackTag*,
		const **CpaCyDhPhase2SecretKeyGenOpData** *	*pPhase2SecretKeyGenData*,
		CpaFlatBuffer *	*pOctetStringSecretKey*
	)		

File: cpa_cy_dh.h

Function to implement Diffie-Hellman phase 2 operations.

This function may be used to implement the Diffie-Hellman phase 2 operation as defined in the PKCS #3 standard. It may be used to generate the Diffie-Hellman shared secret key.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pDhPhase2Cb*	Pointer to a callback function to be invoked when the operation is complete. If the pointer is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pPhase2SecretKeyGenData*	Structure containing all the data needed to perform the DH Phase 2 secret key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pOctetStringSecretKey*	Pointer to memory allocated by the client into which the octet string secret key will be written. The size of this buffer in bytes (as represented by the dataLenInBytes field) MUST be at least big enough to store the public value, which may have

a bit length up to that of pPrimeP. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pDhPhase2Cb is non-NULL an asynchronous callback of type CpaCyGenFlatBufCbFunc is generated in response to this function call. Any errors generated during processing are reported in the structure returned in the callback.

See also:
CpaCyGenFlatBufCbFunc, **CpaCyDhPhase2SecretKeyGenOpData**

CpaStatus CPA_DEPRECATED cpaCyDhQueryStats (const **CpaInstanceHandle** *instanceHandle*,
struct **_CpaCyDhStats** * *pDhStats*
)

File: cpa_cy_dh.h

Query statistics for Diffie-Hellman operations

Deprecated:
As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyDhQueryStats64()**.

This function will query a specific Instance handle for Diffie- Hellman statistics. The user MUST allocate the CpaCyDhStats structure and pass the reference to that structure into this function call. This function writes the statistic results into the passed in CpaCyDhStats structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[out]	*pDhStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
CpaCyDhStats

CpaStatus cpaCyDhQueryStats64 (const **CpaInstanceHandle** *instanceHandle*,
CpaCyDhStats64 * *pDhStats*
)

File: cpa_cy_dh.h

Query statistics (64-bit version) for Diffie-Hellman operations

This function will query a specific Instance handle for the 64-bit version of the Diffie-Hellman statistics. The user MUST allocate the CpaCyDhStats64 structure and pass the reference to that structure into this function call. This function writes the statistic results into the passed in CpaCyDhStats64 structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[out]`	*pDhStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
CpaCyDhStats64

13 Digital Signature Algorithm (DSA) API

[Cryptographic API]

Collaboration diagram for Digital Signature Algorithm (DSA) API:

13.1 Detailed Description

File: cpa_cy_dsa.h

These functions specify the API for Public Key Encryption (Cryptography) Digital Signature Algorithm (DSA) operations.

Support is provided for FIPS PUB 186-2 with Change Notice 1 specification, and optionally for FIPS PUB 186-3. If an implementation does not support FIPS PUB 186-3, then the corresponding functions may return a status of **CPA_STATUS_FAIL**.

Support for FIPS PUB 186-2 with Change Notice 1 implies supporting the following choice for the pair L and N:

- L = 1024, N = 160

Support for FIPS PUB 186-3 implies supporting the following choices for the pair L and N:

- L = 1024, N = 160
- L = 2048, N = 224
- L = 2048, N = 256
- L = 3072, N = 256

Only the modular math aspects of DSA parameter generation and message signature generation and verification are implemented here. For full DSA support, this DSA API SHOULD be used in conjunction with other parts of this overall Cryptographic API. In particular the Symmetric functions (for hashing), the Random Number Generation functions, and the Prime Number Test functions will be required.

Note:

Large numbers are represented on the QuickAssist API as described in the Large Number API (**Cryptographic Large Number API**).

13.2 Data Structures

- struct **_CpaCyDsaPParamGenOpData**
- struct **_CpaCyDsaGParamGenOpData**
- struct **_CpaCyDsaYParamGenOpData**
- struct **_CpaCyDsaRSignOpData**
- struct **_CpaCyDsaSSignOpData**
- struct **_CpaCyDsaRSSignOpData**
- struct **_CpaCyDsaVerifyOpData**
- struct **_CpaCyDsaStats**
- struct **_CpaCyDsaStats64**

13.3 Typedefs

- typedef **_CpaCyDsaPParamGenOpData CpaCyDsaPParamGenOpData**
- typedef **_CpaCyDsaGParamGenOpData CpaCyDsaGParamGenOpData**
- typedef **_CpaCyDsaYParamGenOpData CpaCyDsaYParamGenOpData**
- typedef **_CpaCyDsaRSignOpData CpaCyDsaRSignOpData**
- typedef **_CpaCyDsaSSignOpData CpaCyDsaSSignOpData**
- typedef **_CpaCyDsaRSSignOpData CpaCyDsaRSSignOpData**
- typedef **_CpaCyDsaVerifyOpData CpaCyDsaVerifyOpData**
- typedef **_CpaCyDsaStats CPA_DEPRECATED**
- typedef **_CpaCyDsaStats64 CpaCyDsaStats64**
- typedef void(* **CpaCyDsaGenCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** protocolStatus, **CpaFlatBuffer** *pOut)
- typedef void(* **CpaCyDsaRSSignCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** protocolStatus, **CpaFlatBuffer** *pR, **CpaFlatBuffer** *pS)
- typedef void(* **CpaCyDsaVerifyCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** verifyStatus)

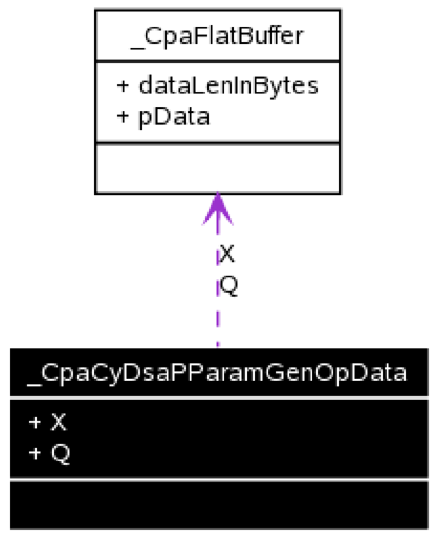
13.4 Functions

- **CpaStatus cpaCyDsaGenPParam** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaGenCbFunc** pCb, void *pCallbackTag, const **CpaCyDsaPParamGenOpData** *pOpData, **CpaBoolean** *pProtocolStatus, **CpaFlatBuffer** *pP)
- **CpaStatus cpaCyDsaGenGParam** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaGenCbFunc** pCb, void *pCallbackTag, const **CpaCyDsaGParamGenOpData** *pOpData, **CpaBoolean** *pProtocolStatus, **CpaFlatBuffer** *pG)
- **CpaStatus cpaCyDsaGenYParam** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaGenCbFunc** pCb, void *pCallbackTag, const **CpaCyDsaYParamGenOpData** *pOpData, **CpaBoolean** *pProtocolStatus, **CpaFlatBuffer** *pY)
- **CpaStatus cpaCyDsaSignR** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaGenCbFunc** pCb, void *pCallbackTag, const **CpaCyDsaRSignOpData** *pOpData, **CpaBoolean** *pProtocolStatus, **CpaFlatBuffer** *pR)
- **CpaStatus cpaCyDsaSignS** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaGenCbFunc** pCb, void *pCallbackTag, const **CpaCyDsaSSignOpData** *pOpData, **CpaBoolean** *pProtocolStatus, **CpaFlatBuffer** *pS)
- **CpaStatus cpaCyDsaSignRS** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaRSSignCbFunc** pCb, void *pCallbackTag, const **CpaCyDsaRSSignOpData** *pOpData, **CpaBoolean** *pProtocolStatus, **CpaFlatBuffer** *pR, **CpaFlatBuffer** *pS)
- **CpaStatus cpaCyDsaVerify** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaVerifyCbFunc** pCb, void *pCallbackTag, const **CpaCyDsaVerifyOpData** *pOpData, **CpaBoolean** *pVerifyStatus)
- **CpaStatus CPA_DEPRECATED cpaCyDsaQueryStats** (const **CpaInstanceHandle** instanceHandle, struct **_CpaCyDsaStats** *pDsaStats)
- **CpaStatus cpaCyDsaQueryStats64** (const **CpaInstanceHandle** instanceHandle, **CpaCyDsaStats64** *pDsaStats)

13.5 Data Structure Documentation

13.5.1 _CpaCyDsaPParamGenOpData Struct Reference

Collaboration diagram for _CpaCyDsaPParamGenOpData:



13.5.1.1 Detailed Description

File: cpa_cy_dsa.h

DSA P Parameter Generation Operation Data.

This structure contains the operation data for the cpaCyDsaGenPParam function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. X.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaGenPParam function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyDsaGenPParam()

13.5.1.2 Data Fields

- **CpaFlatBuffer X**
- **CpaFlatBuffer Q**

13.5.1.3 Field Documentation

CpaFlatBuffer _CpaCyDsaPParamGenOpData::X

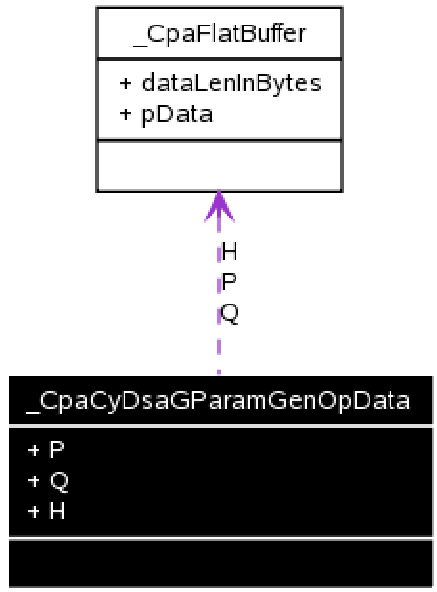
2^(L-1) <= X < 2^L (from FIPS 186-3)

CpaFlatBuffer _CpaCyDsaPParamGenOpData::Q

DSA group parameter q

13.5.2 _CpaCyDsaGParamGenOpData Struct Reference

Collaboration diagram for _CpaCyDsaGParamGenOpData:



13.5.2.1 Detailed Description

File: cpa_cy_dsa.h

DSA G Parameter Generation Operation Data.

This structure contains the operation data for the cpaCyDsaGenGParam function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

All numbers MUST be stored in big-endian order.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaGenGParam function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyDsaGenGParam()

13.5.2.2 Data Fields

- **CpaFlatBuffer P**
- **CpaFlatBuffer Q**
- **CpaFlatBuffer H**

13.5.2.3 Field Documentation

CpaFlatBuffer _CpaCyDsaGParamGenOpData::P

DSA group parameter p

CpaFlatBuffer _CpaCyDsaGParamGenOpData::Q

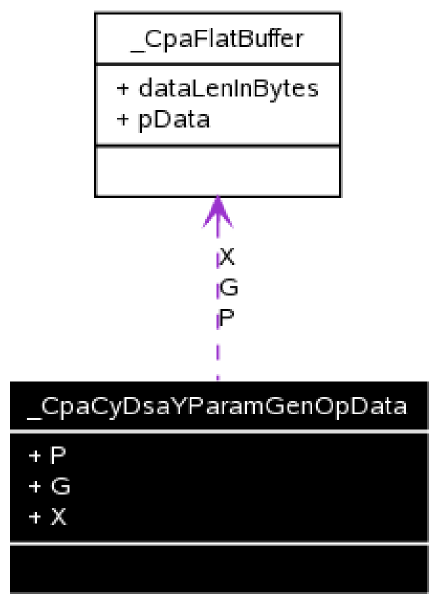
DSA group parameter q

CpaFlatBuffer _CpaCyDsaGParamGenOpData::H

any integer with 1 < h < p - 1

13.5.3 _CpaCyDsaYParamGenOpData Struct Reference

Collaboration diagram for _CpaCyDsaYParamGenOpData:



13.5.3.1 Detailed Description

File: cpa_cy_dsa.h

DSA Y Parameter Generation Operation Data.

This structure contains the operation data for the cpaCyDsaGenYParam function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaGenYParam function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyDsaGenYParam()

13.5.3.2 Data Fields

- **CpaFlatBuffer P**
- **CpaFlatBuffer G**
- **CpaFlatBuffer X**

13.5.3.3 Field Documentation

CpaFlatBuffer _CpaCyDsaYParamGenOpData::P

DSA group parameter p

CpaFlatBuffer _CpaCyDsaYParamGenOpData::G

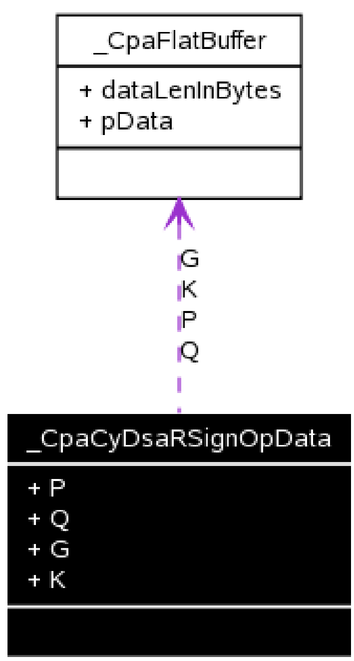
DSA group parameter g

CpaFlatBuffer _CpaCyDsaYParamGenOpData::X

DSA private key x

13.5.4 _CpaCyDsaRSignOpData Struct Reference

Collaboration diagram for _CpaCyDsaRSignOpData:



13.5.4.1 Detailed Description

File: cpa_cy_dsa.h

DSA R Sign Operation Data.

This structure contains the operation data for the cpaCyDsaSignR function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaSignR function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyDsaSignR()

13.5.4.2 Data Fields

- **CpaFlatBuffer P**
- **CpaFlatBuffer Q**
- **CpaFlatBuffer G**
- **CpaFlatBuffer K**

13.5.4.3 Field Documentation

CpaFlatBuffer _CpaCyDsaRSignOpData::P

DSA group parameter p

CpaFlatBuffer _CpaCyDsaRSignOpData::Q

DSA group parameter q

CpaFlatBuffer _CpaCyDsaRSignOpData::G

DSA group parameter g

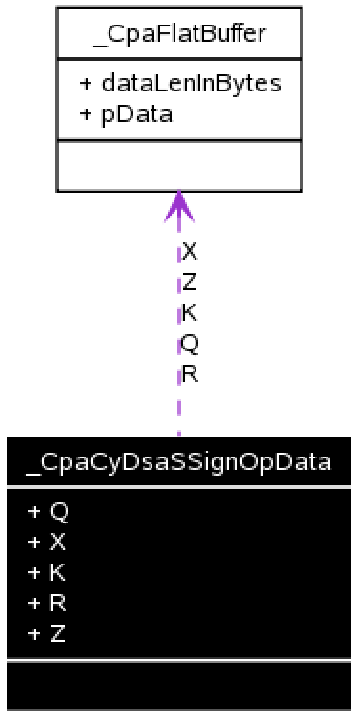
CpaFlatBuffer _CpaCyDsaRSignOpData::K

DSA secret parameter k for signing

13.5.5 _CpaCyDsaSSignOpData Struct Reference

Collaboration diagram for _CpaCyDsaSSignOpData:

13.5.5 _CpaCyDsaSSignOpData Struct Reference



13.5.5.1 Detailed Description

File: cpa_cy_dsa.h

DSA S Sign Operation Data.

This structure contains the operation data for the cpaCyDsaSignS function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. Q.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaSignS function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyDsaSignS()

13.5.5.2 Data Fields

- **CpaFlatBuffer Q**
- **CpaFlatBuffer X**
- **CpaFlatBuffer K**
- **CpaFlatBuffer R**
- **CpaFlatBuffer Z**

13.5.5.3 Field Documentation

CpaFlatBuffer _CpaCyDsaSSignOpData::Q

DSA group parameter q

CpaFlatBuffer _CpaCyDsaSSignOpData::X

DSA private key x

CpaFlatBuffer _CpaCyDsaSSignOpData::K

DSA secret parameter k for signing

CpaFlatBuffer _CpaCyDsaSSignOpData::R

DSA message signature r

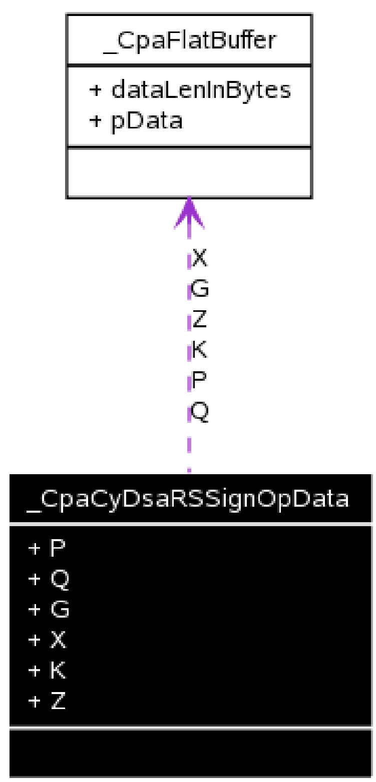
CpaFlatBuffer _CpaCyDsaSSignOpData::Z

The leftmost min(N, outlen) bits of Hash(M), where:

- N is the bit length of q
- outlen is the bit length of the hash function output block
- M is the message to be signed

13.5.6 _CpaCyDsaRSSignOpData Struct Reference

Collaboration diagram for _CpaCyDsaRSSignOpData:



13.5.6 _CpaCyDsaRSSignOpData Struct Reference

13.5.6.1 Detailed Description

File: cpa_cy_dsa.h

DSA R & S Sign Operation Data.

This structure contains the operation data for the cpaCyDsaSignRS function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaSignRS function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyDsaSignRS()

13.5.6.2 Data Fields

- **CpaFlatBuffer P**
- **CpaFlatBuffer Q**
- **CpaFlatBuffer G**
- **CpaFlatBuffer X**
- **CpaFlatBuffer K**
- **CpaFlatBuffer Z**

13.5.6.3 Field Documentation

CpaFlatBuffer _CpaCyDsaRSSignOpData::P

DSA group parameter p

CpaFlatBuffer _CpaCyDsaRSSignOpData::Q

DSA group parameter q

CpaFlatBuffer _CpaCyDsaRSSignOpData::G

DSA group parameter g

CpaFlatBuffer _CpaCyDsaRSSignOpData::X

DSA private key x

CpaFlatBuffer _CpaCyDsaRSSignOpData::K

DSA secret parameter k for signing

CpaFlatBuffer _CpaCyDsaRSSignOpData::Z

The leftmost min(N, outlen) bits of Hash(M), where:

- N is the bit length of q

- outlen is the bit length of the hash function output block
- M is the message to be signed

13.5.7 _CpaCyDsaVerifyOpData Struct Reference

Collaboration diagram for _CpaCyDsaVerifyOpData:

13.5.7.1 Detailed Description

File: cpa_cy_dsa.h

DSA Verify Operation Data.

This structure contains the operation data for the cpaCyDsaVerify function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaVerify function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyDsaVerify()

13.5.7.2 Data Fields

- **CpaFlatBuffer P**
- **CpaFlatBuffer Q**
- **CpaFlatBuffer G**
- **CpaFlatBuffer Y**
- **CpaFlatBuffer Z**
- **CpaFlatBuffer R**
- **CpaFlatBuffer S**

13.5.7.3 Field Documentation

CpaFlatBuffer _CpaCyDsaVerifyOpData::P

DSA group parameter p

CpaFlatBuffer _CpaCyDsaVerifyOpData::Q

DSA group parameter q

CpaFlatBuffer _CpaCyDsaVerifyOpData::G

DSA group parameter g

CpaFlatBuffer _CpaCyDsaVerifyOpData::Y

DSA public key y

CpaFlatBuffer _CpaCyDsaVerifyOpData::Z

The leftmost min(N, outlen) bits of Hash(M'), where:

- N is the bit length of q
- outlen is the bit length of the hash function output block
- M is the message to be signed

CpaFlatBuffer _CpaCyDsaVerifyOpData::R

DSA message signature r

CpaFlatBuffer _CpaCyDsaVerifyOpData::S

DSA message signature s

13.5.8 _CpaCyDsaStats Struct Reference

13.5.8.1 Detailed Description

File: cpa_cy_dsa.h

Cryptographic DSA Statistics.

Deprecated:
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyDsaStats64**.

13.5.8 _CpaCyDsaStats Struct Reference

This structure contains statistics on the Cryptographic DSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

13.5.8.2 Data Fields

- **Cpa32U numDsaPParamGenRequests**
- **Cpa32U numDsaPParamGenRequestErrors**
- **Cpa32U numDsaPParamGenCompleted**
- **Cpa32U numDsaPParamGenCompletedErrors**
- **Cpa32U numDsaGParamGenRequests**
- **Cpa32U numDsaGParamGenRequestErrors**
- **Cpa32U numDsaGParamGenCompleted**
- **Cpa32U numDsaGParamGenCompletedErrors**
- **Cpa32U numDsaYParamGenRequests**
- **Cpa32U numDsaYParamGenRequestErrors**
- **Cpa32U numDsaYParamGenCompleted**
- **Cpa32U numDsaYParamGenCompletedErrors**
- **Cpa32U numDsaRSignRequests**
- **Cpa32U numDsaRSignRequestErrors**
- **Cpa32U numDsaRSignCompleted**
- **Cpa32U numDsaRSignCompletedErrors**
- **Cpa32U numDsaSSignRequests**
- **Cpa32U numDsaSSignRequestErrors**
- **Cpa32U numDsaSSignCompleted**
- **Cpa32U numDsaSSignCompletedErrors**
- **Cpa32U numDsaRSSignRequests**
- **Cpa32U numDsaRSSignRequestErrors**
- **Cpa32U numDsaRSSignCompleted**
- **Cpa32U numDsaRSSignCompletedErrors**
- **Cpa32U numDsaVerifyRequests**
- **Cpa32U numDsaVerifyRequestErrors**
- **Cpa32U numDsaVerifyCompleted**
- **Cpa32U numDsaVerifyCompletedErrors**
- **Cpa32U numDsaVerifyFailures**

13.5.8.3 Field Documentation

Cpa32U _CpaCyDsaStats::numDsaPParamGenRequests

Total number of successful DSA P parameter generation requests.

Cpa32U _CpaCyDsaStats::numDsaPParamGenRequestErrors

Total number of DSA P parameter generation requests that had an error and could not be processed.

Cpa32U _CpaCyDsaStats::numDsaPParamGenCompleted

Total number of DSA P parameter generation operations that completed successfully.

Cpa32U _CpaCyDsaStats::numDsaPParamGenCompletedErrors

Total number of DSA P parameter generation operations that could not be completed successfully due to errors.

Cpa32U _CpaCyDsaStats::numDsaGParamGenRequests

Total number of successful DSA G parameter generation requests.

Cpa32U _CpaCyDsaStats::numDsaGParamGenRequestErrors

Total number of DSA G parameter generation requests that had an error and could not be processed.

Cpa32U _CpaCyDsaStats::numDsaGParamGenCompleted

Total number of DSA G parameter generation operations that completed successfully.

Cpa32U _CpaCyDsaStats::numDsaGParamGenCompletedErrors

Total number of DSA G parameter generation operations that could not be completed successfully due to errors.

Cpa32U _CpaCyDsaStats::numDsaYParamGenRequests

Total number of successful DSA Y parameter generation requests.

Cpa32U _CpaCyDsaStats::numDsaYParamGenRequestErrors

Total number of DSA Y parameter generation requests that had an error and could not be processed.

Cpa32U _CpaCyDsaStats::numDsaYParamGenCompleted

Total number of DSA Y parameter generation operations that completed successfully.

Cpa32U _CpaCyDsaStats::numDsaYParamGenCompletedErrors

Total number of DSA Y parameter generation operations that could not be completed successfully due to errors.

Cpa32U _CpaCyDsaStats::numDsaRSignRequests

Total number of successful DSA R sign generation requests.

Cpa32U _CpaCyDsaStats::numDsaRSignRequestErrors

Total number of DSA R sign requests that had an error and could not be processed.

Cpa32U _CpaCyDsaStats::numDsaRSignCompleted

Total number of DSA R sign operations that completed successfully.

Cpa32U _CpaCyDsaStats::numDsaRSignCompletedErrors

Total number of DSA R sign operations that could not be completed successfully due to errors.

Cpa32U _CpaCyDsaStats::numDsaSSignRequests

Total number of successful DSA S sign generation requests.

Cpa32U _CpaCyDsaStats::numDsaSSignRequestErrors

Total number of DSA S sign requests that had an error and could not be processed.

Cpa32U _CpaCyDsaStats::numDsaSSignCompleted

Total number of DSA S sign operations that completed successfully.

Cpa32U _CpaCyDsaStats::numDsaSSignCompletedErrors

Total number of DSA S sign operations that could not be completed successfully due to errors.

Cpa32U _CpaCyDsaStats::numDsaRSSignRequests

Total number of successful DSA RS sign generation requests.

Cpa32U _CpaCyDsaStats::numDsaRSSignRequestErrors

Total number of DSA RS sign requests that had an error and could not be processed.

Cpa32U _CpaCyDsaStats::numDsaRSSignCompleted

Total number of DSA RS sign operations that completed successfully.

Cpa32U _CpaCyDsaStats::numDsaRSSignCompletedErrors

Total number of DSA RS sign operations that could not be completed successfully due to errors.

Cpa32U _CpaCyDsaStats::numDsaVerifyRequests

Total number of successful DSA verify generation requests.

Cpa32U _CpaCyDsaStats::numDsaVerifyRequestErrors

Total number of DSA verify requests that had an error and could not be processed.

Cpa32U _CpaCyDsaStats::numDsaVerifyCompleted

Total number of DSA verify operations that completed successfully.

Cpa32U _CpaCyDsaStats::numDsaVerifyCompletedErrors

Total number of DSA verify operations that could not be completed successfully due to errors.

Cpa32U _CpaCyDsaStats::numDsaVerifyFailures

Total number of DSA verify operations that executed successfully but the outcome of the test was that the verification failed. Note that this does not indicate an error.

13.5.9 _CpaCyDsaStats64 Struct Reference

13.5.9.1 Detailed Description

File: cpa_cy_dsa.h

Cryptographic DSA Statistics (64-bit version).

This structure contains 64-bit version of the statistics on the Cryptographic DSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

13.5.9.2 Data Fields

- **Cpa64U numDsaPParamGenRequests**
- **Cpa64U numDsaPParamGenRequestErrors**
- **Cpa64U numDsaPParamGenCompleted**
- **Cpa64U numDsaPParamGenCompletedErrors**
- **Cpa64U numDsaGParamGenRequests**
- **Cpa64U numDsaGParamGenRequestErrors**
- **Cpa64U numDsaGParamGenCompleted**
- **Cpa64U numDsaGParamGenCompletedErrors**
- **Cpa64U numDsaYParamGenRequests**
- **Cpa64U numDsaYParamGenRequestErrors**
- **Cpa64U numDsaYParamGenCompleted**
- **Cpa64U numDsaYParamGenCompletedErrors**
- **Cpa64U numDsaRSignRequests**
- **Cpa64U numDsaRSignRequestErrors**

- **Cpa64U numDsaRSignCompleted**
- **Cpa64U numDsaRSignCompletedErrors**
- **Cpa64U numDsaSSignRequests**
- **Cpa64U numDsaSSignRequestErrors**
- **Cpa64U numDsaSSignCompleted**
- **Cpa64U numDsaSSignCompletedErrors**
- **Cpa64U numDsaRSSignRequests**
- **Cpa64U numDsaRSSignRequestErrors**
- **Cpa64U numDsaRSSignCompleted**
- **Cpa64U numDsaRSSignCompletedErrors**
- **Cpa64U numDsaVerifyRequests**
- **Cpa64U numDsaVerifyRequestErrors**
- **Cpa64U numDsaVerifyCompleted**
- **Cpa64U numDsaVerifyCompletedErrors**
- **Cpa64U numDsaVerifyFailures**

13.5.9.3 Field Documentation

Cpa64U _CpaCyDsaStats64::numDsaPParamGenRequests

Total number of successful DSA P parameter generation requests.

Cpa64U _CpaCyDsaStats64::numDsaPParamGenRequestErrors

Total number of DSA P parameter generation requests that had an error and could not be processed.

Cpa64U _CpaCyDsaStats64::numDsaPParamGenCompleted

Total number of DSA P parameter generation operations that completed successfully.

Cpa64U _CpaCyDsaStats64::numDsaPParamGenCompletedErrors

Total number of DSA P parameter generation operations that could not be completed successfully due to errors.

Cpa64U _CpaCyDsaStats64::numDsaGParamGenRequests

Total number of successful DSA G parameter generation requests.

Cpa64U _CpaCyDsaStats64::numDsaGParamGenRequestErrors

Total number of DSA G parameter generation requests that had an error and could not be processed.

Cpa64U _CpaCyDsaStats64::numDsaGParamGenCompleted

Total number of DSA G parameter generation operations that completed successfully.

Cpa64U _CpaCyDsaStats64::numDsaGParamGenCompletedErrors

Total number of DSA G parameter generation operations that could not be completed successfully due to errors.

Cpa64U _CpaCyDsaStats64::numDsaYParamGenRequests

Total number of successful DSA Y parameter generation requests.

Cpa64U _CpaCyDsaStats64::numDsaYParamGenRequestErrors

Total number of DSA Y parameter generation requests that had an error and could not be processed.

Cpa64U _CpaCyDsaStats64::numDsaYParamGenCompleted

Total number of DSA Y parameter generation operations that completed successfully.

Cpa64U _CpaCyDsaStats64::numDsaYParamGenCompletedErrors

Total number of DSA Y parameter generation operations that could not be completed successfully due to errors.

Cpa64U _CpaCyDsaStats64::numDsaRSignRequests

Total number of successful DSA R sign generation requests.

Cpa64U _CpaCyDsaStats64::numDsaRSignRequestErrors

Total number of DSA R sign requests that had an error and could not be processed.

Cpa64U _CpaCyDsaStats64::numDsaRSignCompleted

Total number of DSA R sign operations that completed successfully.

Cpa64U _CpaCyDsaStats64::numDsaRSignCompletedErrors

Total number of DSA R sign operations that could not be completed successfully due to errors.

Cpa64U _CpaCyDsaStats64::numDsaSSignRequests

Total number of successful DSA S sign generation requests.

Cpa64U _CpaCyDsaStats64::numDsaSSignRequestErrors

Total number of DSA S sign requests that had an error and could not be processed.

Cpa64U _CpaCyDsaStats64::numDsaSSignCompleted

Total number of DSA S sign operations that completed successfully.

Cpa64U _CpaCyDsaStats64::numDsaSSignCompletedErrors

Total number of DSA S sign operations that could not be completed successfully due to errors.

Cpa64U _CpaCyDsaStats64::numDsaRSSignRequests

Total number of successful DSA RS sign generation requests.

Cpa64U _CpaCyDsaStats64::numDsaRSSignRequestErrors

Total number of DSA RS sign requests that had an error and could not be processed.

Cpa64U _CpaCyDsaStats64::numDsaRSSignCompleted

Total number of DSA RS sign operations that completed successfully.

Cpa64U _CpaCyDsaStats64::numDsaRSSignCompletedErrors

Total number of DSA RS sign operations that could not be completed successfully due to errors.

Cpa64U _CpaCyDsaStats64::numDsaVerifyRequests

Total number of successful DSA verify generation requests.

Cpa64U _CpaCyDsaStats64::numDsaVerifyRequestErrors

Total number of DSA verify requests that had an error and could not be processed.

Cpa64U _CpaCyDsaStats64::numDsaVerifyCompleted

Total number of DSA verify operations that completed successfully.

Cpa64U _CpaCyDsaStats64::numDsaVerifyCompletedErrors

Total number of DSA verify operations that could not be completed successfully due to errors.

Cpa64U _CpaCyDsaStats64::numDsaVerifyFailures

Total number of DSA verify operations that executed successfully but the outcome of the test was that the verification failed. Note that this does not indicate an error.

13.6 Typedef Documentation

typedef struct **_CpaCyDsaPParamGenOpData CpaCyDsaPParamGenOpData**

File: cpa_cy_dsa.h

DSA P Parameter Generation Operation Data.

This structure contains the operation data for the cpaCyDsaGenPParam function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. X.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaGenPParam function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyDsaGenPParam()

typedef struct **_CpaCyDsaGParamGenOpData CpaCyDsaGParamGenOpData**

File: cpa_cy_dsa.h

DSA G Parameter Generation Operation Data.

This structure contains the operation data for the cpaCyDsaGenGParam function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

All numbers MUST be stored in big-endian order.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaGenGParam function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyDsaGenGParam()

typedef struct **_CpaCyDsaYParamGenOpData CpaCyDsaYParamGenOpData**

File: cpa_cy_dsa.h

DSA Y Parameter Generation Operation Data.

This structure contains the operation data for the cpaCyDsaGenYParam function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaGenYParam function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyDsaGenYParam()

typedef struct **_CpaCyDsaRSignOpData CpaCyDsaRSignOpData**

File: cpa_cy_dsa.h

DSA R Sign Operation Data.

This structure contains the operation data for the cpaCyDsaSignR function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaSignR function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyDsaSignR()

typedef struct **_CpaCyDsaSSignOpData CpaCyDsaSSignOpData**

File: cpa_cy_dsa.h

DSA S Sign Operation Data.

This structure contains the operation data for the cpaCyDsaSignS function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. Q.pData[0] = MSB.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaSignS function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyDsaSignS()

typedef struct **_CpaCyDsaRSSignOpData CpaCyDsaRSSignOpData**

File: cpa_cy_dsa.h

DSA R & S Sign Operation Data.

This structure contains the operation data for the cpaCyDsaSignRS function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaSignRS function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyDsaSignRS()

typedef struct **_CpaCyDsaVerifyOpData CpaCyDsaVerifyOpData**

File: cpa_cy_dsa.h

DSA Verify Operation Data.

This structure contains the operation data for the cpaCyDsaVerify function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyDsaVerify function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyDsaVerify()

typedef struct **_CpaCyDsaStats CPA_DEPRECATED**

File: cpa_cy_dsa.h

Cryptographic DSA Statistics.

Deprecated:
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyDsaStats64**.

This structure contains statistics on the Cryptographic DSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef struct **_CpaCyDsaStats64 CpaCyDsaStats64**

File: cpa_cy_dsa.h

Cryptographic DSA Statistics (64-bit version).

This structure contains 64-bit version of the statistics on the Cryptographic DSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef void(* **CpaCyDsaGenCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** protocolStatus, **CpaFlatBuffer** *pOut)

File: cpa_cy_dsa.h

Definition of a generic callback function invoked for a number of the DSA API functions..

This is the prototype for the cpaCyDsaGenCbFunc callback function.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:
`[in]` *pCallbackTag* User-supplied value to help identify request.

[in]	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
[in]	*pOpData*	Opaque pointer to Operation data supplied in request.
[in]	*protocolStatus*	The result passes/fails the DSA protocol related checks.
[in]	*pOut*	Output data from the request.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
cpaCyDsaGenPParam() cpaCyDsaGenGParam() cpaCyDsaSignR() cpaCyDsaSignS()

typedef void(* **CpaCyDsaRSSignCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** protocolStatus, **CpaFlatBuffer** *pR, **CpaFlatBuffer** *pS)

File: cpa_cy_dsa.h

Definition of callback function invoked for cpaCyDsaSignRS requests.

This is the prototype for the cpaCyDsaSignRS callback function, which will provide the DSA message signature r and s parameters.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
[in]	*pOpData*	Operation data pointer supplied in request.
[in]	*protocolStatus*	The result passes/fails the DSA protocol related checks.
[in]	*pR*	DSA message signature r.
[in]	*pS*	DSA message signature s.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
cpaCyDsaSignRS()

typedef void(* **CpaCyDsaVerifyCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** verifyStatus)

File: cpa_cy_dsa.h

Definition of callback function invoked for cpaCyDsaVerify requests.

This is the prototype for the cpaCyDsaVerify callback function.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*pCallbackTag*	User-supplied value to help identify request.
`[in]`	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
`[in]`	*pOpData*	Operation data pointer supplied in request.
`[in]`	*verifyStatus*	The verification passed or failed.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
cpaCyDsaVerify()

13.7 Function Documentation

CpaStatus cpaCyDsaGenPParam (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyDsaGenCbFunc**	*pCb*,
	void *	*pCallbackTag*,
	const **CpaCyDsaPParamGenOpData** *	*pOpData*,
	CpaBoolean *	*pProtocolStatus*,
	CpaFlatBuffer *	*pP*
)		

File: cpa_cy_dsa.h

Generate DSA P Parameter.

This function performs FIPS 186-3 Appendix A.1.1.2 steps 11.4 and 11.5, and part of step 11.7:

11.4. c = X mod 2q. 11.5. p = X - (c - 1). 11.7. Test whether or not p is prime as specified in Appendix C.3. [Note that a GCD test against ~1400 small primes is performed on p to eliminate ~94% of composites - this is NOT a "robust" primality test, as specified in Appendix C.3.]

The protocol status, returned in the callback function as parameter protocolStatus (or, in the case of synchronous invocation, in the parameter *pProtocolStatus) is used to indicate whether the value p is in the right range and has passed the limited primality test.

Specifically, (protocolStatus == CPA_TRUE) means p is in the right range and SHOULD be subjected to a robust primality test as specified in FIPS 186-3 Appendix C.3 (for example, 40 rounds of Miller-Rabin). Meanwhile, (protocolStatus == CPA_FALSE) means p is either composite, or p < 2^(L-1), in which case the value of p gets set to zero.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pProtocolStatus*	The result passes/fails the DSA protocol related checks.
[out]	*pP*	Candidate for DSA parameter p, p odd and 2^(L-1) < p < X On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized.

Postcondition:

None

Note:

When pCb is non-NULL an asynchronous callback of type CpaCyDsaPParamGenCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:

CpaCyDsaPParamGenOpData, **CpaCyDsaGenCbFunc**

```
CpaStatus cpaCyDsaGenGParam ( const CpaInstanceHandle          instanceHandle,
                              const CpaCyDsaGenCbFunc          pCb,
                              void *                           pCallbackTag,
                              const CpaCyDsaGParamGenOpData *  pOpData,
                              CpaBoolean *                     pProtocolStatus,
                              CpaFlatBuffer *                  pG
                            )
```

File: cpa_cy_dsa.h

Generate DSA G Parameter.

This function performs FIPS 186-3 Appendix A.2.1, steps 1 and 3, and part of step 4:

1. e = (p - 1)/q. 3. Set g = h^e mod p. 4. If (g = 1), then go to step 2. Here, the implementation will check for g == 1, and return status accordingly.

The protocol status, returned in the callback function as parameter protocolStatus (or, in the case of synchronous invocation, in the parameter *pProtocolStatus) is used to indicate whether the value g is acceptable.

Specifically, (protocolStatus == CPA_TRUE) means g is acceptable. Meanwhile, (protocolStatus == CPA_FALSE) means g == 1, so a different value of h SHOULD be used to generate another value of g.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pProtocolStatus*	The result passes/fails the DSA protocol related checks.
[out]	*pG*	g = h^((p-1)/q) mod p. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pCb is non-NULL an asynchronous callback of type CpaCyDsaGParamGenCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be

8-byte aligned.

See also:
CpaCyDsaGParamGenOpData, **CpaCyDsaGenCbFunc**

CpaStatus cpaCyDsaGenYParam (	const **CpaInstanceHandle**		*instanceHandle*,
	const **CpaCyDsaGenCbFunc**		*pCb*,
	void *		*pCallbackTag*,
	const **CpaCyDsaYParamGenOpData** *		*pOpData*,
	CpaBoolean *		*pProtocolStatus*,
	CpaFlatBuffer *		*pY*
)			

File: cpa_cy_dsa.h

Generate DSA Y Parameter.

This function performs modular exponentiation to generate y as described in FIPS 186-3 section 4.1: y = g^x mod p

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pProtocolStatus*	The result passes/fails the DSA protocol related checks.
[out]	*pY*	y = g^x mod p* On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.

CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized via cpaCyStartInstance function.

Postcondition:

None

Note:

When pCb is non-NULL an asynchronous callback of type CpaCyDsaYParamGenCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:

CpaCyDsaYParamGenOpData, **CpaCyDsaGenCbFunc**

```
CpaStatus cpaCyDsaSignR ( const CpaInstanceHandle        instanceHandle,
                          const CpaCyDsaGenCbFunc        pCb,
                          void *                         pCallbackTag,
                          const CpaCyDsaRSignOpData *    pOpData,
                          CpaBoolean *                   pProtocolStatus,
                          CpaFlatBuffer *                pR
                        )
```

File: cpa_cy_dsa.h

Generate DSA R Signature.

This function generates the DSA R signature as described in FIPS 186-3 Section 4.6: r = (g^k mod p) mod q

The protocol status, returned in the callback function as parameter protocolStatus (or, in the case of synchronous invocation, in the parameter *pProtocolStatus) is used to indicate whether the value r == 0.

Specifically, (protocolStatus == CPA_TRUE) means r != 0, while (protocolStatus == CPA_FALSE) means r == 0.

Generation of signature r does not depend on the content of the message being signed, so this operation can be done in advance for different values of k. Then once each message becomes available only the signature s needs to be generated.

Context:

When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:

None

Side-Effects:

None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pProtocolStatus*	The result passes/fails the DSA protocol related checks.
[out]	*pR*	DSA message signature r. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pCb is non-NULL an asynchronous callback of type CpaCyDsaRSignCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyDsaRSignOpData, **CpaCyDsaGenCbFunc**, **cpaCyDsaSignS()**, **cpaCyDsaSignRS()**

CpaStatus cpaCyDsaSignS (const **CpaInstanceHandle** *instanceHandle*,
const **CpaCyDsaGenCbFunc** *pCb*,
void * *pCallbackTag*,
const **CpaCyDsaSSignOpData** * *pOpData*,
CpaBoolean * *pProtocolStatus*,
CpaFlatBuffer * *pS*
)

File: cpa_cy_dsa.h

Generate DSA S Signature.

This function generates the DSA S signature as described in FIPS 186-3 Section 4.6: s = (k^-1(z + xr)) mod q

Here, z = the leftmost min(N, outlen) bits of Hash(M). This function does not perform the SHA digest; z is computed by the caller and passed as a parameter in the pOpData field.

The protocol status, returned in the callback function as parameter protocolStatus (or, in the case of synchronous invocation, in the parameter *pProtocolStatus) is used to indicate whether the value s == 0.

Specifically, (protocolStatus == CPA_TRUE) means s != 0, while (protocolStatus == CPA_FALSE) means s == 0.

If signature r has been generated in advance, then this function can be used to generate the signature s once the message becomes available.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	User-supplied value to help identify request.
`[in]`	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pProtocolStatus*	The result passes/fails the DSA protocol related checks.
`[out]`	*pS*	DSA message signature s. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.

CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pCb is non-NULL an asynchronous callback of type CpaCyDsaSSignCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyDsaSSignOpData, **CpaCyDsaGenCbFunc**, **cpaCyDsaSignR()**, **cpaCyDsaSignRS()**

CpaStatus cpaCyDsaSignRS (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyDsaRSSignCbFunc**	*pCb*,
	void *	*pCallbackTag*,
	const **CpaCyDsaRSSignOpData** *	*pOpData*,
	CpaBoolean *	*pProtocolStatus*,
	CpaFlatBuffer *	*pR*,
	CpaFlatBuffer *	*pS*
)		

File: cpa_cy_dsa.h

Generate DSA R and S Signatures.

This function generates the DSA R and S signatures as described in FIPS 186-3 Section 4.6:

r = (g^k mod p) mod q s = (k^-1(z + xr)) mod q

Here, z = the leftmost min(N, outlen) bits of Hash(M). This function does not perform the SHA digest; z is computed by the caller and passed as a parameter in the pOpData field.

The protocol status, returned in the callback function as parameter protocolStatus (or, in the case of synchronous invocation, in the parameter *pProtocolStatus) is used to indicate whether either of the values r or s are zero.

Specifically, (protocolStatus == CPA_TRUE) means neither is zero (i.e. (r != 0) && (s != 0)), while (protocolStatus == CPA_FALSE) means that at least one of r or s is zero (i.e. (r == 0) || (s == 0)).

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pProtocolStatus*	The result passes/fails the DSA protocol related checks.
[out]	*pR*	DSA message signature r.
[out]	*pS*	DSA message signature s.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pCb is non-NULL an asynchronous callback of type CpaCyDsaRSSignCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyDsaRSSignOpData, **CpaCyDsaRSSignCbFunc**, **cpaCyDsaSignR()**, **cpaCyDsaSignS()**

CpaStatus cpaCyDsaVerify (const **CpaInstanceHandle** *instanceHandle*,
const **CpaCyDsaVerifyCbFunc** *pCb*,
void * *pCallbackTag*,
const **CpaCyDsaVerifyOpData** * *pOpData*,
CpaBoolean * *pVerifyStatus*
)

File: cpa_cy_dsa.h

Verify DSA R and S signatures.

This function performs FIPS 186-3 Section 4.7: w = (s')^-1 mod q u1 = (zw) mod q u2 = ((r')w) mod q v = (((g)^u1 (y)^u2) mod p) mod q

Here, z = the leftmost min(N, outlen) bits of Hash(M'). This function does not perform the SHA digest; z is computed by the caller and passed as a parameter in the pOpData field.

A response status of ok (verifyStatus == CPA_TRUE) means v = r'. A response status of not ok (verifyStatus == CPA_FALSE) means v != r'.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	User-supplied value to help identify request.
`[in]`	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pVerifyStatus*	The verification passed or failed.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pCb is non-NULL an asynchronous callback of type CpaCyDsaVerifyCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyDsaVerifyOpData, **CpaCyDsaVerifyCbFunc**

CpaStatus CPA_DEPRECATED cpaCyDsaQueryStats (	const **CpaInstanceHandle**	*instanceHandle*,
	struct **_CpaCyDsaStats** *	*pDsaStats*
)		

File: cpa_cy_dsa.h

Query statistics for a specific DSA instance.

Deprecated:
As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyDsaQueryStats64()**.

This function will query a specific instance of the DSA implementation for statistics. The user MUST allocate the CpaCyDsaStats structure and pass the reference to that structure into this function call. This function writes the statistic results into the passed in CpaCyDsaStats structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[out]	*pDsaStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.

CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
CpaCyDsaStats

CpaStatus cpaCyDsaQueryStats64 (const **CpaInstanceHandle** *instanceHandle*, **CpaCyDsaStats64** * *pDsaStats*)

File: cpa_cy_dsa.h

Query 64-bit statistics for a specific DSA instance.

This function will query a specific instance of the DSA implementation for 64-bit statistics. The user MUST allocate the CpaCyDsaStats64 structure and pass the reference to that structure into this function. This function writes the statistic results into the passed in CpaCyDsaStats64 structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[out]`	*pDsaStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

Component has been initialized.

Postcondition:

None

Note:

This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:

CpaCyDsaStats

14 Elliptic Curve (EC) API

[Cryptographic API]

Collaboration diagram for Elliptic Curve (EC) API:

14.1 Detailed Description

File: cpa_cy_ec.h

These functions specify the API for Public Key Encryption (Cryptography) Elliptic Curve (EC) operations.

All implementations will support at least the following:

- "NIST RECOMMENDED ELLIPTIC CURVES FOR FEDERAL GOVERNMENT USE" as defined by http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf

- Random curves where the max(log2(q), log2(n) + log2(h)) <= 512 where q is the modulus, n is the order of the curve and h is the cofactor

Note:

Large numbers are represented on the QuickAssist API as described in the Large Number API (**Cryptographic Large Number API**).

In addition, the bit length of large numbers passed to the API MUST NOT exceed 576 bits for Elliptic Curve operations.

14.2 Data Structures

- struct **_CpaCyEcPointMultiplyOpData**
- struct **_CpaCyEcPointVerifyOpData**
- struct **_CpaCyEcStats64**

14.3 Typedefs

- typedef enum **_CpaCyEcFieldType CpaCyEcFieldType**
- typedef **_CpaCyEcPointMultiplyOpData CpaCyEcPointMultiplyOpData**
- typedef **_CpaCyEcPointVerifyOpData CpaCyEcPointVerifyOpData**
- typedef **_CpaCyEcStats64 CpaCyEcStats64**
- typedef void(* **CpaCyEcPointMultiplyCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** multiplyStatus, **CpaFlatBuffer** *pXk, **CpaFlatBuffer** *pYk)
- typedef void(* **CpaCyEcPointVerifyCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** verifyStatus)

14.4 Enumerations

- enum **_CpaCyEcFieldType** {
  CPA_CY_EC_FIELD_TYPE_PRIME,

CPA_CY_EC_FIELD_TYPE_BINARY
}

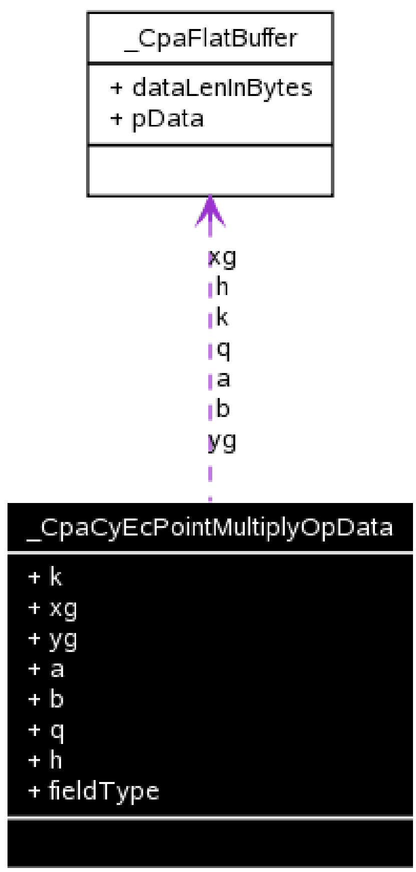
14.5 Functions

- **CpaStatus cpaCyEcPointMultiply** (const **CpaInstanceHandle** instanceHandle, const **CpaCyEcPointMultiplyCbFunc** pCb, void *pCallbackTag, const **CpaCyEcPointMultiplyOpData** *pOpData, **CpaBoolean** *pMultiplyStatus, **CpaFlatBuffer** *pXk, **CpaFlatBuffer** *pYk)
- **CpaStatus cpaCyEcPointVerify** (const **CpaInstanceHandle** instanceHandle, const **CpaCyEcPointVerifyCbFunc** pCb, void *pCallbackTag, const **CpaCyEcPointVerifyOpData** *pOpData, **CpaBoolean** *pVerifyStatus)
- **CpaStatus cpaCyEcQueryStats64** (const **CpaInstanceHandle** instanceHandle, **CpaCyEcStats64** *pEcStats)
- **CpaStatus cpaCyKptEcPointMultiply** (const **CpaInstanceHandle** instanceHandle, const **CpaCyEcPointMultiplyCbFunc** pCb, void *pCallbackTag, const **CpaCyEcPointMultiplyOpData** *pOpData, **CpaBoolean** *pMultiplyStatus, **CpaFlatBuffer** *pXk, **CpaFlatBuffer** *pYk, **CpaFlatBuffer** *pKptUnwrapContext)

14.6 Data Structure Documentation

14.6.1 _CpaCyEcPointMultiplyOpData Struct Reference

Collaboration diagram for _CpaCyEcPointMultiplyOpData:



14.6.1 _CpaCyEcPointMultiplyOpData Struct Reference

14.6.1.1 Detailed Description

File: cpa_cy_ec.h

EC Point Multiplication Operation Data.

This structure contains the operation data for the cpaCyEcPointMultiply function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcPointMultiply function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyEcPointMultiply()

14.6.1.2 Data Fields

- **CpaFlatBuffer k**
- **CpaFlatBuffer xg**
- **CpaFlatBuffer yg**
- **CpaFlatBuffer a**
- **CpaFlatBuffer b**
- **CpaFlatBuffer q**
- **CpaFlatBuffer h**
- **CpaCyEcFieldType fieldType**

14.6.1.3 Field Documentation

CpaFlatBuffer _CpaCyEcPointMultiplyOpData::k

scalar multiplier (k > 0 and k < n)

CpaFlatBuffer _CpaCyEcPointMultiplyOpData::xg

x coordinate of curve point

CpaFlatBuffer _CpaCyEcPointMultiplyOpData::yg

y coordinate of curve point

CpaFlatBuffer _CpaCyEcPointMultiplyOpData::a

a elliptic curve coefficient

CpaFlatBuffer _CpaCyEcPointMultiplyOpData::b

b elliptic curve coefficient

CpaFlatBuffer _CpaCyEcPointMultiplyOpData::q

prime modulus or irreducible polynomial over GF(2^m)

CpaFlatBuffer _CpaCyEcPointMultiplyOpData::h

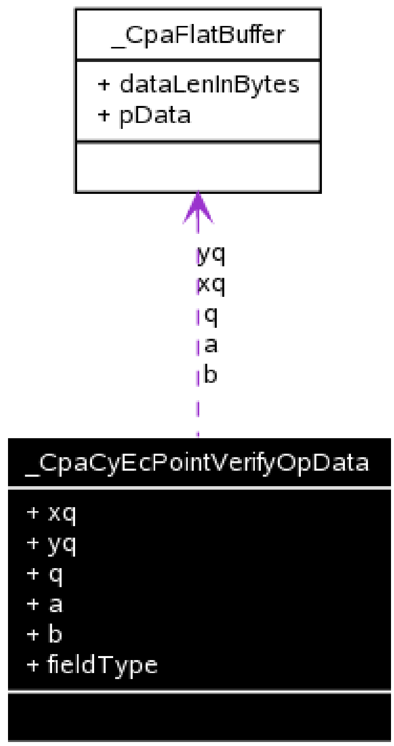
cofactor of the operation. If the cofactor is NOT required then set the cofactor to 1 or the data pointer of the Flat Buffer to NULL.

CpaCyEcFieldType _CpaCyEcPointMultiplyOpData::fieldType

field type for the operation

14.6.2 _CpaCyEcPointVerifyOpData Struct Reference

Collaboration diagram for _CpaCyEcPointVerifyOpData:



14.6.2.1 Detailed Description

File: cpa_cy_ec.h

EC Point Verification Operation Data.

This structure contains the operation data for the cpaCyEcPointVerify function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the CpaCyEcPointVerify function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyEcPointVerify()

14.6.2.2 Data Fields

- **CpaFlatBuffer xq**
- **CpaFlatBuffer yq**
- **CpaFlatBuffer q**
- **CpaFlatBuffer a**
- **CpaFlatBuffer b**
- **CpaCyEcFieldType fieldType**

14.6.2.3 Field Documentation

CpaFlatBuffer _CpaCyEcPointVerifyOpData::xq

x coordinate candidate point

CpaFlatBuffer _CpaCyEcPointVerifyOpData::yq

y coordinate candidate point

CpaFlatBuffer _CpaCyEcPointVerifyOpData::q

prime modulus or irreducible polynomial over GF(2^m)

CpaFlatBuffer _CpaCyEcPointVerifyOpData::a

a elliptic curve coefficient

CpaFlatBuffer _CpaCyEcPointVerifyOpData::b

b elliptic curve coefficient

CpaCyEcFieldType _CpaCyEcPointVerifyOpData::fieldType

field type for the operation

14.6.3 _CpaCyEcStats64 Struct Reference

14.6.3.1 Detailed Description

File: cpa_cy_ec.h

Cryptographic EC Statistics.

This structure contains statistics on the Cryptographic EC operations. Statistics are set to zero when the component is initialized, and are collected per instance.

14.6.3.2 Data Fields

- **Cpa64U numEcPointMultiplyRequests**
- **Cpa64U numEcPointMultiplyRequestErrors**
- **Cpa64U numEcPointMultiplyCompleted**
- **Cpa64U numEcPointMultiplyCompletedError**
- **Cpa64U numEcPointMultiplyCompletedOutputInvalid**
- **Cpa64U numEcPointVerifyRequests**
- **Cpa64U numEcPointVerifyRequestErrors**

- **Cpa64U numEcPointVerifyCompleted**
- **Cpa64U numEcPointVerifyCompletedErrors**
- **Cpa64U numEcPointVerifyCompletedOutputInvalid**

14.6.3.3 Field Documentation

Cpa64U _CpaCyEcStats64::numEcPointMultiplyRequests

Total number of EC Point Multiplication operation requests.

Cpa64U _CpaCyEcStats64::numEcPointMultiplyRequestErrors

Total number of EC Point Multiplication operation requests that had an error and could not be processed.

Cpa64U _CpaCyEcStats64::numEcPointMultiplyCompleted

Total number of EC Point Multiplication operation requests that completed successfully.

Cpa64U _CpaCyEcStats64::numEcPointMultiplyCompletedError

Total number of EC Point Multiplication operation requests that could not be completed successfully due to errors.

Cpa64U _CpaCyEcStats64::numEcPointMultiplyCompletedOutputInvalid

Total number of EC Point Multiplication operation requests that could not be completed successfully due to an invalid output. Note that this does not indicate an error.

Cpa64U _CpaCyEcStats64::numEcPointVerifyRequests

Total number of EC Point Verification operation requests.

Cpa64U _CpaCyEcStats64::numEcPointVerifyRequestErrors

Total number of EC Point Verification operation requests that had an error and could not be processed.

Cpa64U _CpaCyEcStats64::numEcPointVerifyCompleted

Total number of EC Point Verification operation requests that completed successfully.

Cpa64U _CpaCyEcStats64::numEcPointVerifyCompletedErrors

Total number of EC Point Verification operation requests that could not be completed successfully due to errors.

Cpa64U _CpaCyEcStats64::numEcPointVerifyCompletedOutputInvalid

Total number of EC Point Verification operation requests that had an invalid output. Note that this does not indicate an error.

14.7 Typedef Documentation

typedef enum **_CpaCyEcFieldType CpaCyEcFieldType**

File: cpa_cy_ec.h

Field types for Elliptic Curve

As defined by FIPS-186-3, for each cryptovariable length, there are two kinds of fields.

- A prime field is the field GF(p) which contains a prime number p of elements. The elements of this field are the integers modulo p, and the field arithmetic is implemented in terms of the arithmetic of integers modulo p.
- A binary field is the field GF(2^m) which contains 2^m elements for some m (called the degree of the field). The elements of this field are the bit strings of length m, and the field arithmetic is implemented in terms of operations on the bits.

typedef struct **_CpaCyEcPointMultiplyOpData CpaCyEcPointMultiplyOpData**

File: cpa_cy_ec.h

EC Point Multiplication Operation Data.

This structure contains the operation data for the cpaCyEcPointMultiply function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcPointMultiply function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyEcPointMultiply()

typedef struct **_CpaCyEcPointVerifyOpData CpaCyEcPointVerifyOpData**

File: cpa_cy_ec.h

EC Point Verification Operation Data.

This structure contains the operation data for the cpaCyEcPointVerify function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the CpaCyEcPointVerify function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyEcPointVerify()

typedef struct **_CpaCyEcStats64 CpaCyEcStats64**

File: cpa_cy_ec.h

Cryptographic EC Statistics.

This structure contains statistics on the Cryptographic EC operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef void(* **CpaCyEcPointMultiplyCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** multiplyStatus, **CpaFlatBuffer** *pXk, **CpaFlatBuffer** *pYk)

File: cpa_cy_ec.h

Definition of callback function invoked for cpaCyEcPointMultiply requests.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
[in]	*pOpData*	Opaque pointer to Operation data supplied in request.
[in]	*multiplyStatus*	Status of the point multiplication.
[in]	*pXk*	x coordinate of resultant EC point.
[in]	*pYk*	y coordinate of resultant EC point.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
cpaCyEcPointMultiply()

typedef void(* **CpaCyEcPointVerifyCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** verifyStatus)

File: cpa_cy_ec.h

Definition of callback function invoked for cpaCyEcPointVerify requests.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
[in]	*pOpData*	Operation data pointer supplied in request.
[in]	*verifyStatus*	Set to CPA_FALSE if the point is NOT on the curve or at infinity. Set to CPA_TRUE if the point is on the curve.

Returns:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
cpaCyEcPointVerify()

14.8 Enumeration Type Documentation

enum **_CpaCyEcFieldType**

File: cpa_cy_ec.h

Field types for Elliptic Curve

As defined by FIPS-186-3, for each cryptovariable length, there are two kinds of fields.

- A prime field is the field GF(p) which contains a prime number p of elements. The elements of this field are the integers modulo p, and the field arithmetic is implemented in terms of the arithmetic of integers modulo p.

- A binary field is the field GF(2^m) which contains 2^m elements for some m (called the degree of the field). The elements of this field are the bit strings of length m, and the field arithmetic is implemented in terms of operations on the bits.

Enumerator:

CPA_CY_EC_FIELD_TYPE_PRIME	A prime field, GF(p)
CPA_CY_EC_FIELD_TYPE_BINARY	A binary field, GF(2^m)

14.9 Function Documentation

CpaStatus cpaCyEcPointMultiply (	const **CpaInstanceHandle**		*instanceHandle*,
	const **CpaCyEcPointMultiplyCbFunc**		*pCb*,
	void *		*pCallbackTag*,
	const **CpaCyEcPointMultiplyOpData** *		*pOpData*,
	CpaBoolean *		*pMultiplyStatus*,
	CpaFlatBuffer *		*pXk*,
	CpaFlatBuffer *		*pYk*
)			

File: cpa_cy_ec.h

Perform EC Point Multiplication.

This function performs Elliptic Curve Point Multiplication as per ANSI X9.63 Annex D.3.2.

Context:

When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:

None

Side-Effects:

None

Blocking:

Yes when configured to operate in synchronous mode.

Reentrant:

No

Thread-safe:

Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	User-supplied value to help identify request.
`[in]`	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.

[out]	*pMultiplyStatus*	In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
[out]	*pXk*	Pointer to xk flat buffer.
[out]	*pYk*	Pointer to yk flat buffer.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized via cpaCyStartInstance function.

Postcondition:

None

Note:

When pCb is non-NULL an asynchronous callback of type CpaCyEcPointMultiplyCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:

CpaCyEcPointMultiplyOpData, **CpaCyEcPointMultiplyCbFunc**

CpaStatus cpaCyEcPointVerify (const **CpaInstanceHandle** *instanceHandle*,
const **CpaCyEcPointVerifyCbFunc** *pCb*,
void * *pCallbackTag*,
const **CpaCyEcPointVerifyOpData** * *pOpData*,
CpaBoolean * *pVerifyStatus*
)

File: cpa_cy_ec.h

Verify that a point is on an elliptic curve.

This function performs Elliptic Curve Point Verification, as per steps a, b and c of ANSI X9.62 Annex A.4.2. (To perform the final step d, the user can call **cpaCyEcPointMultiply**.)

This function checks if the specified point satisfies the Weierstrass equation for an Elliptic Curve.

For GF(p): y^2 = (x^3 + ax + b) mod p For GF(2^m): y^2 + xy = x^3 + ax^2 + b mod p where p is the irreducible polynomial over GF(2^m)

Use this function to verify a point is in the correct range and is NOT the point at infinity.

Context:

When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:

None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	User-supplied value to help identify request.
`[in]`	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pVerifyStatus*	In synchronous mode, set to CPA_FALSE if the point is NOT on the curve or at infinity. Set to CPA_TRUE if the point is on the curve.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pCb is non-NULL an asynchronous callback of type CpaCyEcPointVerifyCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyEcPointVerifyOpData, **CpaCyEcPointVerifyCbFunc**

CpaStatus cpaCyEcQueryStats64 (const **CpaInstanceHandle** *instanceHandle*,
CpaCyEcStats64 * *pEcStats*
)

File: cpa_cy_ec.h

Query statistics for a specific EC instance.

This function will query a specific instance of the EC implementation for statistics. The user MUST allocate the CpaCyEcStats64 structure and pass the reference to that structure into this function call. This function writes the statistic results into the passed in CpaCyEcStats64 structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[out]`	*pEcStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
CpaCyEcStats64

CpaStatus cpaCyKptEcPointMultiply (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyEcPointMultiplyCbFunc**	*pCb*,
	void *	*pCallbackTag*,
	const **CpaCyEcPointMultiplyOpData** *	*pOpData*,
	CpaBoolean *	*pMultiplyStatus*,
	CpaFlatBuffer *	*pXk*,
	CpaFlatBuffer *	*pYk*,
	CpaFlatBuffer *	*pKptUnwrapContext*
	)	

File: cpa_cy_kpt.h

Perform KPT mode EC Point Multiplication.

This function is variant of cpaCyEcPointMultiply, which will perform Elliptic Curve Point Multiplication as per ANSI X9.63 Annex D.3.2.

Context:

When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:

None

Side-Effects:

None

Blocking:

Yes when configured to operate in synchronous mode.

Reentrant:

No

Thread-safe:

Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pMultiplyStatus*	In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
[out]	*pXk*	Pointer to xk flat buffer.
[out]	*pYk*	Pointer to yk flat buffer.
[in]	*pKptUnwrapContext*	Pointer of structure into which the content of KptUnwrapContext is kept,The client MUST allocate this memory and copy structure KptUnwrapContext into this flat buffer.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
By virtue of invoking the cpaCyKptEcPointMultiply, the implementation understands that CpaCyEcPointMultiplyOpData contains an encrypted private key that requires unwrapping. KptUnwrapContext contains an 'KptHandle' field that points to the unwrapping key in the WKT. When pCb is non-NULL an asynchronous callback of type CpaCyEcPointMultiplyCbFunc is generated in response to this function call. In KPT release,private key field in cpaCyKptEcPointMultiply is a concatenation of cipher text and hash tag. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyEcPointMultiplyOpData, **CpaCyEcPointMultiplyCbFunc**

15 Elliptic Curve Diffie-Hellman (ECDH) API

[Cryptographic API]

Collaboration diagram for Elliptic Curve Diffie-Hellman (ECDH) API:

15.1 Detailed Description

File: cpa_cy_ecdh.h

These functions specify the API for Public Key Encryption (Cryptography) Elliptic Curve Diffie-Hellman (ECDH) operations.

Note:

Large numbers are represented on the QuickAssist API as described in the Large Number API (**Cryptographic Large Number API**).

In addition, the bit length of large numbers passed to the API MUST NOT exceed 576 bits for Elliptic Curve operations.

15.2 Data Structures

- struct **_CpaCyEcdhPointMultiplyOpData**
- struct **_CpaCyEcdhStats64**

15.3 Typedefs

- typedef **_CpaCyEcdhPointMultiplyOpData CpaCyEcdhPointMultiplyOpData**
- typedef **_CpaCyEcdhStats64 CpaCyEcdhStats64**
- typedef void(* **CpaCyEcdhPointMultiplyCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** multiplyStatus, **CpaFlatBuffer** *pXk, **CpaFlatBuffer** *pYk)

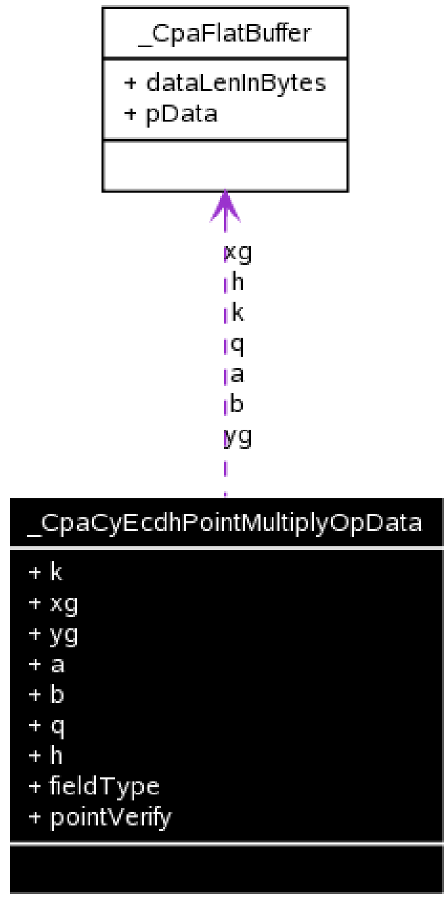
15.4 Functions

- **CpaStatus cpaCyEcdhPointMultiply** (const **CpaInstanceHandle** instanceHandle, const **CpaCyEcdhPointMultiplyCbFunc** pCb, void *pCallbackTag, const **CpaCyEcdhPointMultiplyOpData** *pOpData, **CpaBoolean** *pMultiplyStatus, **CpaFlatBuffer** *pXk, **CpaFlatBuffer** *pYk)
- **CpaStatus cpaCyEcdhQueryStats64** (const **CpaInstanceHandle** instanceHandle, **CpaCyEcdhStats64** *pEcdhStats)

15.5 Data Structure Documentation

15.5.1 _CpaCyEcdhPointMultiplyOpData Struct Reference

Collaboration diagram for _CpaCyEcdhPointMultiplyOpData:



15.5.1.1 Detailed Description

File: cpa_cy_ecdh.h

ECDH Point Multiplication Operation Data.

This structure contains the operation data for the cpaCyEcdhPointMultiply function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcdhPointMultiply function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyEcdhPointMultiply()

15.5.1.2 Data Fields

- **CpaFlatBuffer k**
- **CpaFlatBuffer xg**
- **CpaFlatBuffer yg**
- **CpaFlatBuffer a**
- **CpaFlatBuffer b**
- **CpaFlatBuffer q**
- **CpaFlatBuffer h**
- **CpaCyEcFieldType fieldType**
- **CpaBoolean pointVerify**

15.5.1.3 Field Documentation

CpaFlatBuffer _CpaCyEcdhPointMultiplyOpData::k

scalar multiplier (k > 0 and k < n)

CpaFlatBuffer _CpaCyEcdhPointMultiplyOpData::xg

x coordinate of curve point

CpaFlatBuffer _CpaCyEcdhPointMultiplyOpData::yg

y coordinate of curve point

CpaFlatBuffer _CpaCyEcdhPointMultiplyOpData::a

a equation coefficient

CpaFlatBuffer _CpaCyEcdhPointMultiplyOpData::b

b equation coefficient

CpaFlatBuffer _CpaCyEcdhPointMultiplyOpData::q

prime modulus or irreducible polynomial over GF(2^r)

CpaFlatBuffer _CpaCyEcdhPointMultiplyOpData::h

cofactor of the operation. If the cofactor is NOT required then set the cofactor to 1 or the data pointer of the Flat Buffer to NULL. There are some restrictions on the value of the cofactor. Implementations of this API will support at least the following:

- NIST standard curves and their cofactors (1, 2 and 4)
- Random curves where max(log2(p), log2(n)+log2(h)) <= 512, where p is the modulus, n is the order of the curve and h is the cofactor

CpaCyEcFieldType _CpaCyEcdhPointMultiplyOpData::fieldType

field type for the operation

CpaBoolean _CpaCyEcdhPointMultiplyOpData::pointVerify

set to CPA_TRUE to do a verification before the multiplication

---

15.5.2 _CpaCyEcdhStats64 Struct Reference

15.5.2.1 Detailed Description

File: cpa_cy_ecdh.h

Cryptographic ECDH Statistics.

This structure contains statistics on the Cryptographic ECDH operations. Statistics are set to zero when the component is initialized, and are collected per instance.

15.5.2.2 Data Fields

- **Cpa64U numEcdhPointMultiplyRequests**
- **Cpa64U numEcdhPointMultiplyRequestErrors**
- **Cpa64U numEcdhPointMultiplyCompleted**
- **Cpa64U numEcdhPointMultiplyCompletedError**
- **Cpa64U numEcdhRequestCompletedOutputInvalid**

15.5.2.3 Field Documentation

Cpa64U _CpaCyEcdhStats64::numEcdhPointMultiplyRequests

Total number of ECDH Point Multiplication operation requests.

Cpa64U _CpaCyEcdhStats64::numEcdhPointMultiplyRequestErrors

Total number of ECDH Point Multiplication operation requests that had an error and could not be processed.

Cpa64U _CpaCyEcdhStats64::numEcdhPointMultiplyCompleted

Total number of ECDH Point Multiplication operation requests that completed successfully.

Cpa64U _CpaCyEcdhStats64::numEcdhPointMultiplyCompletedError

Total number of ECDH Point Multiplication operation requests that could not be completed successfully due to errors.

Cpa64U _CpaCyEcdhStats64::numEcdhRequestCompletedOutputInvalid

Total number of ECDH Point Multiplication or Point Verify operation requests that could not be completed successfully due to an invalid output. Note that this does not indicate an error.

15.6 Typedef Documentation

typedef struct **_CpaCyEcdhPointMultiplyOpData CpaCyEcdhPointMultiplyOpData**

File: cpa_cy_ecdh.h

ECDH Point Multiplication Operation Data.

This structure contains the operation data for the cpaCyEcdhPointMultiply function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcdhPointMultiply function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyEcdhPointMultiply()

typedef struct **_CpaCyEcdhStats64 CpaCyEcdhStats64**

File: cpa_cy_ecdh.h

Cryptographic ECDH Statistics.

This structure contains statistics on the Cryptographic ECDH operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef void(* **CpaCyEcdhPointMultiplyCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** multiplyStatus, **CpaFlatBuffer** *pXk, **CpaFlatBuffer** *pYk)

File: cpa_cy_ecdh.h

Definition of callback function invoked for cpaCyEcdhPointMultiply requests.

This is the prototype for the CpaCyEcdhPointMultiplyCbFunc callback function

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
[in]	*pOpData*	Opaque pointer to Operation data supplied in request.
[in]	*pXk*	Output x coordinate from the request.
[in]	*pYk*	Output y coordinate from the request.
[in]	*multiplyStatus*	Status of the point multiplication and the verification when the pointVerify bit is set in the CpaCyEcdhPointMultiplyOpData structure.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
cpaCyEcdhPointMultiply()

15.7 Function Documentation

CpaStatus cpaCyEcdhPointMultiply (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyEcdhPointMultiplyCbFunc**	*pCb*,
	void *	*pCallbackTag*,
	const **CpaCyEcdhPointMultiplyOpData** *	*pOpData*,
	CpaBoolean *	*pMultiplyStatus*,
	CpaFlatBuffer *	*pXk*,
	CpaFlatBuffer *	*pYk*
	)	

File: cpa_cy_ecdh.h

ECDH Point Multiplication.

This function performs ECDH Point Multiplication as defined in ANSI X9.63 2001 section 5.4

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	User-supplied value to help identify request.

`[in]`	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pMultiplyStatus*	In synchronous mode, the status of the point multiplication and the verification when the pointVerify bit is set in the CpaCyEcdhPointMultiplyOpData structure. Set to CPA_FALSE if the point is NOT on the curve or at infinity. Set to CPA_TRUE if the point is on the curve.
`[out]`	*pXk*	Pointer to x coordinate flat buffer.
`[out]`	*pYk*	Pointer to y coordinate flat buffer.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized via cpaCyStartInstance function.

Postcondition:

None

Note:

When pCb is non-NULL an asynchronous callback of type CpaCyEcdhPointMultiplyCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:

CpaCyEcdhPointMultiplyOpData, **CpaCyEcdhPointMultiplyCbFunc**

CpaStatus cpaCyEcdhQueryStats64 (const **CpaInstanceHandle** *instanceHandle*, **CpaCyEcdhStats64** * *pEcdhStats*)

File: cpa_cy_ecdh.h

Query statistics for a specific ECDH instance.

This function will query a specific instance of the ECDH implementation for statistics. The user MUST allocate the CpaCyEcdhStats64 structure and pass the reference to that structure into this function call. This function writes the statistic results into the passed in CpaCyEcdhStats64 structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:

This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:

None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[out]	*pEcdhStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
CpaCyEcdhStats64

16 Elliptic Curve Digital Signature Algorithm (ECDSA) API

[Cryptographic API]

Collaboration diagram for Elliptic Curve Digital Signature Algorithm (ECDSA) API:

16.1 Detailed Description

File: cpa_cy_ecdsa.h

These functions specify the API for Public Key Encryption (Cryptography) Elliptic Curve Digital Signature Algorithm (ECDSA) operations.

Note:

Large numbers are represented on the QuickAssist API as described in the Large Number API (**Cryptographic Large Number API**).

In addition, the bit length of large numbers passed to the API MUST NOT exceed 576 bits for Elliptic Curve operations.

16.2 Data Structures

- struct **_CpaCyEcdsaSignROpData**
- struct **_CpaCyEcdsaSignSOpData**
- struct **_CpaCyEcdsaSignRSOpData**
- struct **_CpaCyEcdsaVerifyOpData**
- struct **_CpaCyEcdsaStats64**

16.3 Typedefs

- typedef **_CpaCyEcdsaSignROpData CpaCyEcdsaSignROpData**
- typedef **_CpaCyEcdsaSignSOpData CpaCyEcdsaSignSOpData**
- typedef **_CpaCyEcdsaSignRSOpData CpaCyEcdsaSignRSOpData**
- typedef **_CpaCyEcdsaVerifyOpData CpaCyEcdsaVerifyOpData**
- typedef **_CpaCyEcdsaStats64 CpaCyEcdsaStats64**
- typedef void(* **CpaCyEcdsaGenSignCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** multiplyStatus, **CpaFlatBuffer** *pOut)
- typedef void(* **CpaCyEcdsaSignRSCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** multiplyStatus, **CpaFlatBuffer** *pR, **CpaFlatBuffer** *pS)
- typedef void(* **CpaCyEcdsaVerifyCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** verifyStatus)

16.4 Functions

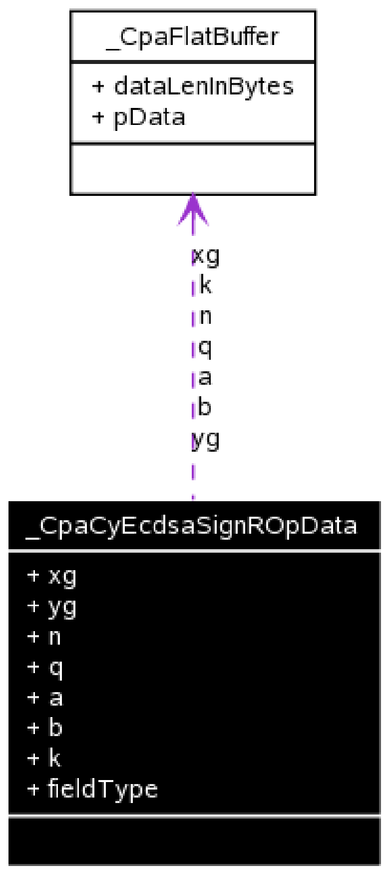
- **CpaStatus cpaCyEcdsaSignR** (const **CpaInstanceHandle** instanceHandle, const **CpaCyEcdsaGenSignCbFunc** pCb, void *pCallbackTag, const **CpaCyEcdsaSignROpData** *pOpData, **CpaBoolean** *pSignStatus, **CpaFlatBuffer** *pR)
- **CpaStatus cpaCyEcdsaSignS** (const **CpaInstanceHandle** instanceHandle, const

CpaCyEcdsaGenSignCbFunc pCb, void *pCallbackTag, const **CpaCyEcdsaSignSOpData** *pOpData, **CpaBoolean** *pSignStatus, **CpaFlatBuffer** *pS)
- **CpaStatus cpaCyEcdsaSignRS** (const **CpaInstanceHandle** instanceHandle, const **CpaCyEcdsaSignRSCbFunc** pCb, void *pCallbackTag, const **CpaCyEcdsaSignRSOpData** *pOpData, **CpaBoolean** *pSignStatus, **CpaFlatBuffer** *pR, **CpaFlatBuffer** *pS)
- **CpaStatus cpaCyEcdsaVerify** (const **CpaInstanceHandle** instanceHandle, const **CpaCyEcdsaVerifyCbFunc** pCb, void *pCallbackTag, const **CpaCyEcdsaVerifyOpData** *pOpData, **CpaBoolean** *pVerifyStatus)
- **CpaStatus cpaCyEcdsaQueryStats64** (const **CpaInstanceHandle** instanceHandle, **CpaCyEcdsaStats64** *pEcdsaStats)

16.5 Data Structure Documentation

16.5.1 _CpaCyEcdsaSignROpData Struct Reference

Collaboration diagram for _CpaCyEcdsaSignROpData:



16.5.1.1 Detailed Description

File: cpa_cy_ecdsa.h

ECDSA Sign R Operation Data.

This structure contains the operation data for the cpaCyEcdsaSignR function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcdsaSignR function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyEcdsaSignR()

16.5.1.2 Data Fields

- **CpaFlatBuffer xg**
- **CpaFlatBuffer yg**
- **CpaFlatBuffer n**
- **CpaFlatBuffer q**
- **CpaFlatBuffer a**
- **CpaFlatBuffer b**
- **CpaFlatBuffer k**
- **CpaCyEcFieldType fieldType**

16.5.1.3 Field Documentation

CpaFlatBuffer _CpaCyEcdsaSignROpData::xg

x coordinate of base point G

CpaFlatBuffer _CpaCyEcdsaSignROpData::yg

y coordinate of base point G

CpaFlatBuffer _CpaCyEcdsaSignROpData::n

order of the base point G, which shall be prime

CpaFlatBuffer _CpaCyEcdsaSignROpData::q

prime modulus or irreducible polynomial over GF(2^r)

CpaFlatBuffer _CpaCyEcdsaSignROpData::a

a elliptic curve coefficient

CpaFlatBuffer _CpaCyEcdsaSignROpData::b

b elliptic curve coefficient

CpaFlatBuffer _CpaCyEcdsaSignROpData::k

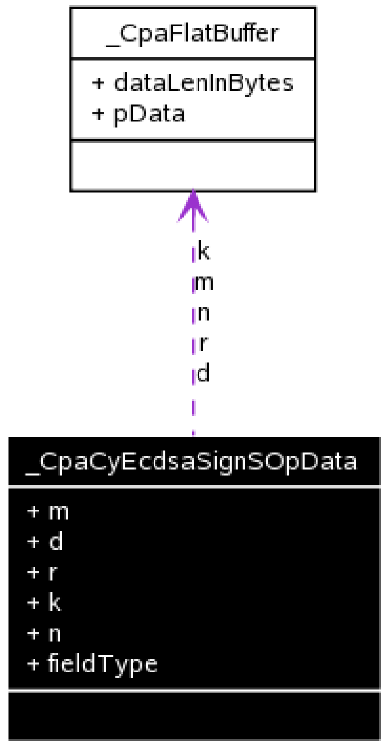
random value (k > 0 and k < n)

CpaCyEcFieldType _CpaCyEcdsaSignROpData::fieldType

field type for the operation

16.5.2 _CpaCyEcdsaSignSOpData Struct Reference

Collaboration diagram for _CpaCyEcdsaSignSOpData:



16.5.2.1 Detailed Description

File: cpa_cy_ecdsa.h

ECDSA Sign S Operation Data.

This structure contains the operation data for the cpaCyEcdsaSignS function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcdsaSignS function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyEcdsaSignS()

16.5.2.2 Data Fields

- **CpaFlatBuffer m**
- **CpaFlatBuffer d**
- **CpaFlatBuffer r**
- **CpaFlatBuffer k**
- **CpaFlatBuffer n**
- **CpaCyEcFieldType fieldType**

16.5.2.3 Field Documentation

CpaFlatBuffer _CpaCyEcdsaSignSOpData::m

digest of the message to be signed

CpaFlatBuffer _CpaCyEcdsaSignSOpData::d

private key

CpaFlatBuffer _CpaCyEcdsaSignSOpData::r

Ecdsa r signature value

CpaFlatBuffer _CpaCyEcdsaSignSOpData::k

random value (k > 0 and k < n)

CpaFlatBuffer _CpaCyEcdsaSignSOpData::n

order of the base point G, which shall be prime

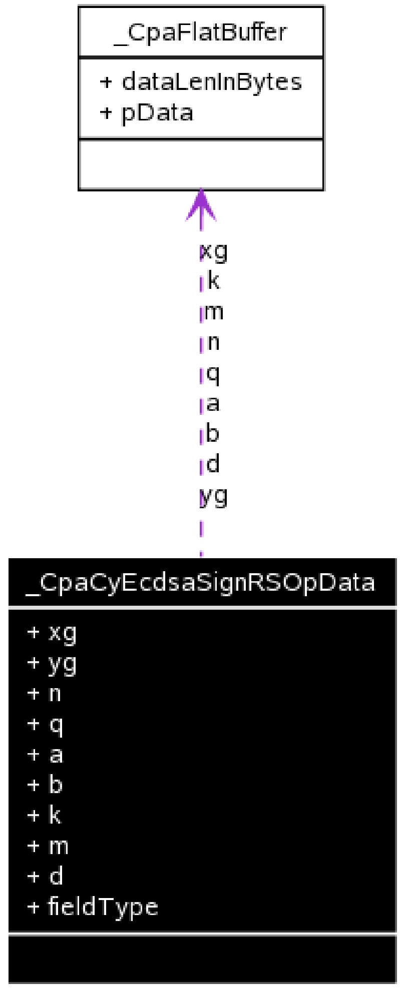
CpaCyEcFieldType _CpaCyEcdsaSignSOpData::fieldType

field type for the operation

16.5.3 _CpaCyEcdsaSignRSOpData Struct Reference

Collaboration diagram for _CpaCyEcdsaSignRSOpData:

16.5.3 _CpaCyEcdsaSignRSOpData Struct Reference



16.5.3.1 Detailed Description

File: cpa_cy_ecdsa.h

ECDSA Sign R & S Operation Data.

This structure contains the operation data for the cpaCyEcdsaSignRS function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcdsaSignRS function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyEcdsaSignRS()

16.5.3.2 Data Fields

- **CpaFlatBuffer xg**
- **CpaFlatBuffer yg**
- **CpaFlatBuffer n**
- **CpaFlatBuffer q**
- **CpaFlatBuffer a**
- **CpaFlatBuffer b**
- **CpaFlatBuffer k**
- **CpaFlatBuffer m**
- **CpaFlatBuffer d**
- **CpaCyEcFieldType fieldType**

16.5.3.3 Field Documentation

CpaFlatBuffer _CpaCyEcdsaSignRSOpData::xg

x coordinate of base point G

CpaFlatBuffer _CpaCyEcdsaSignRSOpData::yg

y coordinate of base point G

CpaFlatBuffer _CpaCyEcdsaSignRSOpData::n

order of the base point G, which shall be prime

CpaFlatBuffer _CpaCyEcdsaSignRSOpData::q

prime modulus or irreducible polynomial over GF(2^r)

CpaFlatBuffer _CpaCyEcdsaSignRSOpData::a

a elliptic curve coefficient

CpaFlatBuffer _CpaCyEcdsaSignRSOpData::b

b elliptic curve coefficient

CpaFlatBuffer _CpaCyEcdsaSignRSOpData::k

random value (k > 0 and k < n)

CpaFlatBuffer _CpaCyEcdsaSignRSOpData::m

digest of the message to be signed

CpaFlatBuffer _CpaCyEcdsaSignRSOpData::d

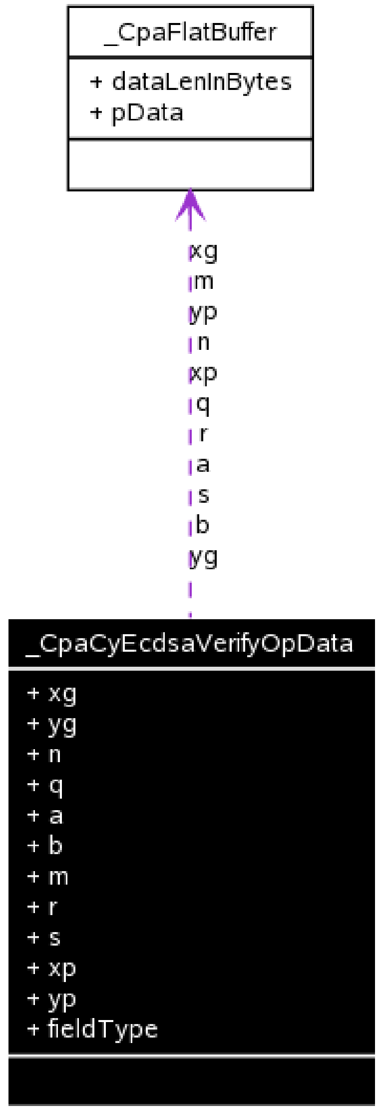
private key

CpaCyEcFieldType _CpaCyEcdsaSignRSOpData::fieldType

field type for the operation

16.5.4 _CpaCyEcdsaVerifyOpData Struct Reference

Collaboration diagram for _CpaCyEcdsaVerifyOpData:



16.5.4.1 Detailed Description

File: cpa_cy_ecdsa.h

ECDSA Verify Operation Data, for Public Key.

This structure contains the operation data for the CpaCyEcdsaVerify function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcdsaVerify function, and before it has been returned in the callback, undefined behavior will result.

See also:

CpaCyEcdsaVerify()

16.5.4.2 Data Fields

- **CpaFlatBuffer xg**
- **CpaFlatBuffer yg**
- **CpaFlatBuffer n**
- **CpaFlatBuffer q**
- **CpaFlatBuffer a**
- **CpaFlatBuffer b**
- **CpaFlatBuffer m**
- **CpaFlatBuffer r**
- **CpaFlatBuffer s**
- **CpaFlatBuffer xp**
- **CpaFlatBuffer yp**
- **CpaCyEcFieldType fieldType**

16.5.4.3 Field Documentation

CpaFlatBuffer _CpaCyEcdsaVerifyOpData::xg

x coordinate of base point G

CpaFlatBuffer _CpaCyEcdsaVerifyOpData::yg

y coordinate of base point G

CpaFlatBuffer _CpaCyEcdsaVerifyOpData::n

order of the base point G, which shall be prime

CpaFlatBuffer _CpaCyEcdsaVerifyOpData::q

prime modulus or irreducible polynomial over GF(2^r)

CpaFlatBuffer _CpaCyEcdsaVerifyOpData::a

a elliptic curve coefficient

CpaFlatBuffer _CpaCyEcdsaVerifyOpData::b

b elliptic curve coefficient

CpaFlatBuffer _CpaCyEcdsaVerifyOpData::m

digest of the message to be signed

CpaFlatBuffer _CpaCyEcdsaVerifyOpData::r

ECDSA r signature value ($r > 0$ and $r < n$)

CpaFlatBuffer _CpaCyEcdsaVerifyOpData::s

ECDSA s signature value ($s > 0$ and $s < n$)

CpaFlatBuffer _CpaCyEcdsaVerifyOpData::xp

x coordinate of point P (public key)

CpaFlatBuffer _CpaCyEcdsaVerifyOpData::yp

y coordinate of point P (public key)

CpaCyEcFieldType _CpaCyEcdsaVerifyOpData::fieldType

field type for the operation

16.5.5 _CpaCyEcdsaStats64 Struct Reference

16.5.5.1 Detailed Description

File: cpa_cy_ecdsa.h

Cryptographic ECDSA Statistics.

This structure contains statistics on the Cryptographic ECDSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

16.5.5.2 Data Fields

- **Cpa64U numEcdsaSignRRequests**
- **Cpa64U numEcdsaSignRRequestErrors**
- **Cpa64U numEcdsaSignRCompleted**
- **Cpa64U numEcdsaSignRCompletedErrors**
- **Cpa64U numEcdsaSignRCompletedOutputInvalid**
- **Cpa64U numEcdsaSignSRequests**
- **Cpa64U numEcdsaSignSRequestErrors**
- **Cpa64U numEcdsaSignSCompleted**
- **Cpa64U numEcdsaSignSCompletedErrors**
- **Cpa64U numEcdsaSignSCompletedOutputInvalid**
- **Cpa64U numEcdsaSignRSRequests**
- **Cpa64U numEcdsaSignRSRequestErrors**
- **Cpa64U numEcdsaSignRSCompleted**
- **Cpa64U numEcdsaSignRSCompletedErrors**
- **Cpa64U numEcdsaSignRSCompletedOutputInvalid**
- **Cpa64U numEcdsaVerifyRequests**
- **Cpa64U numEcdsaVerifyRequestErrors**
- **Cpa64U numEcdsaVerifyCompleted**
- **Cpa64U numEcdsaVerifyCompletedErrors**
- **Cpa64U numEcdsaVerifyCompletedOutputInvalid**

16.5.5.3 Field Documentation

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignRRequests

Total number of ECDSA Sign R operation requests.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignRRequestErrors

Total number of ECDSA Sign R operation requests that had an error and could not be processed.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignRCompleted

Total number of ECDSA Sign R operation requests that completed successfully.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignRCompletedErrors

Total number of ECDSA Sign R operation requests that could not be completed successfully due to errors.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignRCompletedOutputInvalid

Total number of ECDSA Sign R operation requests could not be completed successfully due to an invalid output. Note that this does not indicate an error.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignSRequests

Total number of ECDSA Sign S operation requests.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignSRequestErrors

Total number of ECDSA Sign S operation requests that had an error and could not be processed.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignSCompleted

Total number of ECDSA Sign S operation requests that completed successfully.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignSCompletedErrors

Total number of ECDSA Sign S operation requests that could not be completed successfully due to errors.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignSCompletedOutputInvalid

Total number of ECDSA Sign S operation requests could not be completed successfully due to an invalid output. Note that this does not indicate an error.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignRSRequests

Total number of ECDSA Sign R & S operation requests.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignRSRequestErrors

Total number of ECDSA Sign R & S operation requests that had an error and could not be processed.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignRSCompleted

Total number of ECDSA Sign R & S operation requests that completed successfully.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignRSCompletedErrors

Total number of ECDSA Sign R & S operation requests that could not be completed successfully due to errors.

Cpa64U _CpaCyEcdsaStats64::numEcdsaSignRSCompletedOutputInvalid

Total number of ECDSA Sign R & S operation requests could not be completed successfully due to an invalid output. Note that this does not indicate an error.

Cpa64U _CpaCyEcdsaStats64::numEcdsaVerifyRequests

Total number of ECDSA Verification operation requests.

Cpa64U _CpaCyEcdsaStats64::numEcdsaVerifyRequestErrors

Total number of ECDSA Verification operation requests that had an error and could not be processed.

Cpa64U _CpaCyEcdsaStats64::numEcdsaVerifyCompleted

Total number of ECDSA Verification operation requests that completed successfully.

Cpa64U _CpaCyEcdsaStats64::numEcdsaVerifyCompletedErrors

Total number of ECDSA Verification operation requests that could not be completed successfully due to errors.

Cpa64U _CpaCyEcdsaStats64::numEcdsaVerifyCompletedOutputInvalid

Total number of ECDSA Verification operation requests that resulted in an invalid output. Note that this does not indicate an error.

16.6 Typedef Documentation

typedef struct **_CpaCyEcdsaSignROpData CpaCyEcdsaSignROpData**

File: cpa_cy_ecdsa.h

ECDSA Sign R Operation Data.

This structure contains the operation data for the cpaCyEcdsaSignR function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcdsaSignR function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyEcdsaSignR()

typedef struct **_CpaCyEcdsaSignSOpData CpaCyEcdsaSignSOpData**

File: cpa_cy_ecdsa.h

ECDSA Sign S Operation Data.

This structure contains the operation data for the cpaCyEcdsaSignS function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcdsaSignS function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyEcdsaSignS()

typedef struct **_CpaCyEcdsaSignRSOpData CpaCyEcdsaSignRSOpData**

File: cpa_cy_ecdsa.h

ECDSA Sign R & S Operation Data.

This structure contains the operation data for the cpaCyEcdsaSignRS function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcdsaSignRS function, and before it has been returned in the callback, undefined behavior will result.

See also:
cpaCyEcdsaSignRS()

typedef struct **_CpaCyEcdsaVerifyOpData CpaCyEcdsaVerifyOpData**

File: cpa_cy_ecdsa.h

ECDSA Verify Operation Data, for Public Key.

This structure contains the operation data for the CpaCyEcdsaVerify function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyEcdsaVerify function, and before it has been returned in the callback, undefined behavior will result.

See also:
CpaCyEcdsaVerify()

typedef struct **_CpaCyEcdsaStats64 CpaCyEcdsaStats64**

File: cpa_cy_ecdsa.h

Cryptographic ECDSA Statistics.

This structure contains statistics on the Cryptographic ECDSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef void(* **CpaCyEcdsaGenSignCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** multiplyStatus, **CpaFlatBuffer** *pOut)

File: cpa_cy_ecdsa.h

Definition of a generic callback function invoked for a number of the ECDSA Sign API functions.

This is the prototype for the CpaCyEcdsaGenSignCbFunc callback function.

Context:
: This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
: None

Side-Effects:
: None

Reentrant:
: No

Thread-safe:
: Yes

Parameters:

[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
[in]	*pOpData*	Opaque pointer to Operation data supplied in request.
[in]	*multiplyStatus*	Status of the point multiplication.
[in]	*pOut*	Output data from the request.

Return values:
: *None*

Precondition:
: Component has been initialized.

Postcondition:
: None

Note:
: None

See also:
: **cpaCyEcdsaSignR() cpaCyEcdsaSignS()**

typedef void(* **CpaCyEcdsaSignRSCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** multiplyStatus, **CpaFlatBuffer** *pR, **CpaFlatBuffer** *pS)

File: cpa_cy_ecdsa.h

Definition of callback function invoked for cpaCyEcdsaSignRS requests.

This is the prototype for the CpaCyEcdsaSignRSCbFunc callback function, which will provide the ECDSA message signature r and s parameters.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*pCallbackTag*	User-supplied value to help identify request.
`[in]`	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
`[in]`	*pOpData*	Operation data pointer supplied in request.
`[in]`	*multiplyStatus*	Status of the point multiplication.
`[in]`	*pR*	Ecdsa message signature r.
`[in]`	*pS*	Ecdsa message signature s.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
cpaCyEcdsaSignRS()

typedef void(* **CpaCyEcdsaVerifyCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** verifyStatus)

File: cpa_cy_ecdsa.h

Definition of callback function invoked for cpaCyEcdsaVerify requests.

This is the prototype for the CpaCyEcdsaVerifyCbFunc callback function.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*pCallbackTag*	User-supplied value to help identify request.
`[in]`	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
`[in]`	*pOpData*	Operation data pointer supplied in request.
`[in]`	*verifyStatus*	The verification status.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
cpaCyEcdsaVerify()

16.7 Function Documentation

CpaStatus cpaCyEcdsaSignR	(	const **CpaInstanceHandle**	*instanceHandle*,
		const **CpaCyEcdsaGenSignCbFunc**	*pCb*,
		void *	*pCallbackTag*,
		const **CpaCyEcdsaSignROpData** *	*pOpData*,
		CpaBoolean *	*pSignStatus*,
		CpaFlatBuffer *	*pR*
	)		

File: cpa_cy_ecdsa.h

Generate ECDSA Signature R.

This function generates ECDSA Signature R as per ANSI X9.62 2005 section 7.3.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	User-supplied value to help identify request.
`[in]`	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pSignStatus*	In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
`[out]`	*pR*	ECDSA message signature r.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:

When pCb is non-NULL an asynchronous callback is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
None

CpaStatus cpaCyEcdsaSignS	(	const **CpaInstanceHandle**	*instanceHandle*,
		const **CpaCyEcdsaGenSignCbFunc**	*pCb*,
		void *	*pCallbackTag*,
		const **CpaCyEcdsaSignSOpData** *	*pOpData*,
		CpaBoolean *	*pSignStatus*,
		CpaFlatBuffer *	*pS*
	)		

File: cpa_cy_ecdsa.h

Generate ECDSA Signature S.

This function generates ECDSA Signature S as per ANSI X9.62 2005 section 7.3.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pSignStatus*	In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
[out]	*pS*	ECDSA message signature s.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pCb is non-NULL an asynchronous callback is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
None

CpaStatus cpaCyEcdsaSignRS (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyEcdsaSignRSCbFunc**	*pCb*,
	void *	*pCallbackTag*,
	const **CpaCyEcdsaSignRSOpData** *	*pOpData*,
	CpaBoolean *	*pSignStatus*,
	CpaFlatBuffer *	*pR*,
	CpaFlatBuffer *	*pS*
)		

File: cpa_cy_ecdsa.h

Generate ECDSA Signature R & S.

This function generates ECDSA Signature R & S as per ANSI X9.62 2005 section 7.3.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pSignStatus*	In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
[out]	*pR*	ECDSA message signature r.
[out]	*pS*	ECDSA message signature s.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized via cpaCyStartInstance function.

Postcondition:

None

Note:

When pCb is non-NULL an asynchronous callback is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:

None

CpaStatus cpaCyEcdsaVerify (const **CpaInstanceHandle** *instanceHandle*,
const **CpaCyEcdsaVerifyCbFunc** *pCb*,
void * *pCallbackTag*,
const **CpaCyEcdsaVerifyOpData** * *pOpData*,
CpaBoolean * *pVerifyStatus*
)

File: cpa_cy_ecdsa.h

Verify ECDSA Public Key.

This function performs ECDSA Verify as per ANSI X9.62 2005 section 7.4.

A response status of ok (verifyStatus == CPA_TRUE) means that the signature was verified

Context:

When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pVerifyStatus*	In synchronous mode, set to CPA_FALSE if the point is NOT on the curve or at infinity. Set to CPA_TRUE if the point is on the curve.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pCb is non-NULL an asynchronous callback of type CpaCyEcdsaVerifyCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyEcdsaVerifyOpData, **CpaCyEcdsaVerifyCbFunc**

CpaStatus cpaCyEcdsaQueryStats64 (const **CpaInstanceHandle** *instanceHandle*,
CpaCyEcdsaStats64 * *pEcdsaStats*
)

File: cpa_cy_ecdsa.h

Query statistics for a specific ECDSA instance.

This function will query a specific instance of the ECDSA implementation for statistics. The user MUST allocate the CpaCyEcdsaStats64 structure and pass the reference to that structure into this function call. This function writes the statistic results into the passed in CpaCyEcdsaStats64 structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[out]`	*pEcdsaStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
CpaCyEcdsaStats64

17 Cryptographic Large Number API

[Cryptographic API]

Collaboration diagram for Cryptographic Large Number API:

17.1 Detailed Description

File: cpa_cy_ln.h

These functions specify the Cryptographic API for Large Number Operations.

Note:

Large numbers are represented on the QuickAssist API using octet strings, stored in structures of type **CpaFlatBuffer**. These octet strings are encoded as described by PKCS#1 v2.1, section 4, which is consistent with ASN.1 syntax. The following text summarizes this. Any exceptions to this encoding are specified on the specific data structure or function to which the exception applies.

An n-bit number, N, has a value in the range 2^(n-1) through 2^(n)-1. In other words, its most significant bit, bit n-1 (where bit-counting starts from zero) MUST be set to 1. We can also state that the bit-length n of a number N is defined by n = floor(log2(N))+1.

The buffer, b, in which an n-bit number N is stored, must be "large enough". In other words, b.dataLenInBytes must be at least minLenInBytes = ceiling(n/8).

The number is stored in a "big endian" format. This means that the least significant byte (LSB) is b[b.dataLenInBytes-1], while the most significant byte (MSB) is b[b.dataLenInBytes-minLenInBytes]. In the case where the buffer is "exactly" the right size, then the MSB is b[0]. Otherwise, all bytes from b[0] up to the MSB MUST be set to 0x00.

The largest bit-length we support today is 4096 bits. In other words, we can deal with numbers up to a value of (2^4096)-1.

17.2 Data Structures

- struct **_CpaCyLnModExpOpData**
- struct **_CpaCyLnModInvOpData**
- struct **_CpaCyLnStats**
- struct **_CpaCyLnStats64**

17.3 Typedefs

- typedef **_CpaCyLnModExpOpData CpaCyLnModExpOpData**
- typedef **_CpaCyLnModInvOpData CpaCyLnModInvOpData**
- typedef **_CpaCyLnStats CPA_DEPRECATED**
- typedef **_CpaCyLnStats64 CpaCyLnStats64**

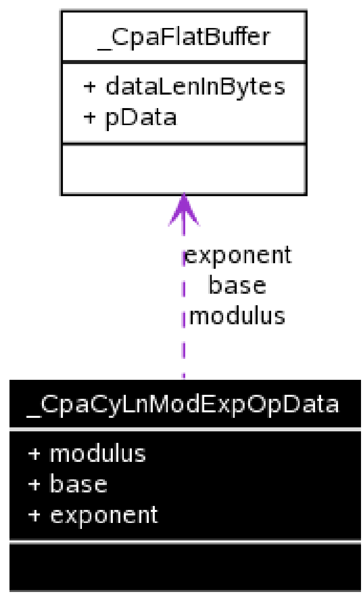
17.4 Functions

- **CpaStatus cpaCyLnModExp** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pLnModExpCb, void *pCallbackTag, const **CpaCyLnModExpOpData** *pLnModExpOpData, **CpaFlatBuffer** *pResult)
- **CpaStatus cpaCyLnModInv** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pLnModInvCb, void *pCallbackTag, const **CpaCyLnModInvOpData** *pLnModInvOpData, **CpaFlatBuffer** *pResult)
- **CpaStatus CPA_DEPRECATED cpaCyLnStatsQuery** (const **CpaInstanceHandle** instanceHandle, struct **_CpaCyLnStats** *pLnStats)
- **CpaStatus cpaCyLnStatsQuery64** (const **CpaInstanceHandle** instanceHandle, **CpaCyLnStats64** *pLnStats)

17.5 Data Structure Documentation

17.5.1 _CpaCyLnModExpOpData Struct Reference

Collaboration diagram for _CpaCyLnModExpOpData:



17.5.1.1 Detailed Description

File: cpa_cy_ln.h

Modular Exponentiation Function Operation Data.

This structure lists the different items that are required in the cpaCyLnModExp function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback. The operation size in bits is equal to the size of whichever of the following is largest: the modulus, the base or the exponent.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyLnModExp function, and before it has been returned in the callback, undefined behavior will result.

The values of the base, the exponent and the modulus MUST all be less than 2^4096, and the modulus must not be equal to zero.

17.5.1.2 Data Fields

- **CpaFlatBuffer modulus**
- **CpaFlatBuffer base**
- **CpaFlatBuffer exponent**

17.5.1.3 Field Documentation

CpaFlatBuffer _CpaCyLnModExpOpData::modulus

Flat buffer containing a pointer to the modulus. This number may be up to 4096 bits in length, and MUST be greater than zero.

CpaFlatBuffer _CpaCyLnModExpOpData::base

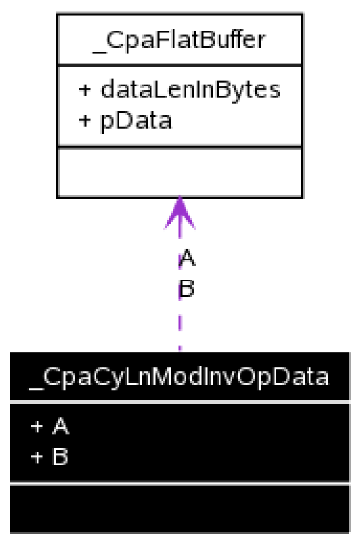
Flat buffer containing a pointer to the base. This number may be up to 4096 bits in length.

CpaFlatBuffer _CpaCyLnModExpOpData::exponent

Flat buffer containing a pointer to the exponent. This number may be up to 4096 bits in length.

17.5.2 _CpaCyLnModInvOpData Struct Reference

Collaboration diagram for _CpaCyLnModInvOpData:



17.5.2.1 Detailed Description

File: cpa_cy_ln.h

Modular Inversion Function Operation Data.

This structure lists the different items that are required in the function **cpaCyLnModInv**. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyLnModInv function, and before it has been returned in the callback, undefined behavior will result.

Note that the values of A and B MUST NOT both be even numbers, and both MUST be less than 2^4096.

17.5.2.2 Data Fields

- **CpaFlatBuffer A**
- **CpaFlatBuffer B**

17.5.2.3 Field Documentation

CpaFlatBuffer _CpaCyLnModInvOpData::A

Flat buffer containing a pointer to the value that will be inverted. This number may be up to 4096 bits in length, it MUST NOT be zero, and it MUST be co-prime with B.

CpaFlatBuffer _CpaCyLnModInvOpData::B

Flat buffer containing a pointer to the value that will be used as the modulus. This number may be up to 4096 bits in length, it MUST NOT be zero, and it MUST be co-prime with A.

17.5.3 _CpaCyLnStats Struct Reference

17.5.3.1 Detailed Description

File: cpa_cy_ln.h

Look Aside Cryptographic large number Statistics.

Deprecated:

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyLnStats64**.

This structure contains statistics on the Look Aside Cryptographic large number operations. Statistics are set to zero when the component is initialized, and are collected per instance.

17.5.3.2 Data Fields

- **Cpa32U numLnModExpRequests**
- **Cpa32U numLnModExpRequestErrors**
- **Cpa32U numLnModExpCompleted**
- **Cpa32U numLnModExpCompletedErrors**
- **Cpa32U numLnModInvRequests**
- **Cpa32U numLnModInvRequestErrors**
- **Cpa32U numLnModInvCompleted**
- **Cpa32U numLnModInvCompletedErrors**

17.5.3.3 Field Documentation

Cpa32U _CpaCyLnStats::numLnModExpRequests

Total number of successful large number modular exponentiation requests.

Cpa32U _CpaCyLnStats::numLnModExpRequestErrors

Total number of large number modular exponentiation requests that had an error and could not be processed.

Cpa32U _CpaCyLnStats::numLnModExpCompleted

Total number of large number modular exponentiation operations that completed successfully.

Cpa32U _CpaCyLnStats::numLnModExpCompletedErrors

Total number of large number modular exponentiation operations that could not be completed successfully due to errors.

Cpa32U _CpaCyLnStats::numLnModInvRequests

Total number of successful large number modular inversion requests.

Cpa32U _CpaCyLnStats::numLnModInvRequestErrors

Total number of large number modular inversion requests that had an error and could not be processed.

Cpa32U _CpaCyLnStats::numLnModInvCompleted

Total number of large number modular inversion operations that completed successfully.

Cpa32U _CpaCyLnStats::numLnModInvCompletedErrors

Total number of large number modular inversion operations that could not be completed successfully due to errors.

17.5.4 _CpaCyLnStats64 Struct Reference

17.5.4.1 Detailed Description

File: cpa_cy_ln.h

Look Aside Cryptographic large number Statistics.

This structure contains statistics on the Look Aside Cryptographic large number operations. Statistics are set to zero when the component is initialized, and are collected per instance.

17.5.4.2 Data Fields

- **Cpa64U numLnModExpRequests**
- **Cpa64U numLnModExpRequestErrors**
- **Cpa64U numLnModExpCompleted**
- **Cpa64U numLnModExpCompletedErrors**
- **Cpa64U numLnModInvRequests**
- **Cpa64U numLnModInvRequestErrors**
- **Cpa64U numLnModInvCompleted**
- **Cpa64U numLnModInvCompletedErrors**

17.5.4.3 Field Documentation

Cpa64U _CpaCyLnStats64::numLnModExpRequests

Total number of successful large number modular exponentiation requests.

Cpa64U _CpaCyLnStats64::numLnModExpRequestErrors

Total number of large number modular exponentiation requests that had an error and could not be processed.

Cpa64U _CpaCyLnStats64::numLnModExpCompleted

Total number of large number modular exponentiation operations that completed successfully.

Cpa64U _CpaCyLnStats64::numLnModExpCompletedErrors

Total number of large number modular exponentiation operations that could not be completed successfully due to errors.

Cpa64U _CpaCyLnStats64::numLnModInvRequests

Total number of successful large number modular inversion requests.

Cpa64U _CpaCyLnStats64::numLnModInvRequestErrors

Total number of large number modular inversion requests that had an error and could not be processed.

Cpa64U _CpaCyLnStats64::numLnModInvCompleted

Total number of large number modular inversion operations that completed successfully.

Cpa64U _CpaCyLnStats64::numLnModInvCompletedErrors

Total number of large number modular inversion operations that could not be completed successfully due to errors.

17.6 Typedef Documentation

typedef struct **_CpaCyLnModExpOpData CpaCyLnModExpOpData**

File: cpa_cy_ln.h

Modular Exponentiation Function Operation Data.

This structure lists the different items that are required in the cpaCyLnModExp function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback. The operation size in bits is equal to the size of whichever of the following is largest: the modulus, the base or the exponent.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyLnModExp function, and before it has been returned in the callback, undefined behavior will result.

The values of the base, the exponent and the modulus MUST all be less than 2^4096, and the modulus must not be equal to zero.

typedef struct **_CpaCyLnModInvOpData CpaCyLnModInvOpData**

File: cpa_cy_ln.h

Modular Inversion Function Operation Data.

This structure lists the different items that are required in the function **cpaCyLnModInv**. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyLnModInv function, and before it has been returned in the callback, undefined behavior will result.

Note that the values of A and B MUST NOT both be even numbers, and both MUST be less than 2^4096.

typedef struct **_CpaCyLnStats CPA_DEPRECATED**

File: cpa_cy_ln.h

Look Aside Cryptographic large number Statistics.

Deprecated:
As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyLnStats64**.

This structure contains statistics on the Look Aside Cryptographic large number operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef struct **_CpaCyLnStats64 CpaCyLnStats64**

File: cpa_cy_ln.h

Look Aside Cryptographic large number Statistics.

This structure contains statistics on the Look Aside Cryptographic large number operations. Statistics are set to zero when the component is initialized, and are collected per instance.

17.7 Function Documentation

CpaStatus cpaCyLnModExp (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyGenFlatBufCbFunc**	*pLnModExpCb*,
	void *	*pCallbackTag*,
	const **CpaCyLnModExpOpData** *	*pLnModExpOpData*,
	CpaFlatBuffer *	*pResult*
	)	

File: cpa_cy_ln.h

Perform modular exponentiation operation.

This function performs modular exponentiation. It computes the following result based on the inputs:

result = (base ^ exponent) mod modulus

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pLnModExpCb*	Pointer to callback function to be invoked when the operation is complete.
[in]	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in]	*pLnModExpOpData*	Structure containing all the data needed to perform the LN modular exponentiation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pResult*	Pointer to a flat buffer containing a pointer to memory allocated by the client into which the result will be written. The size of the memory required MUST be larger than or equal to the size required to store the modulus. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized.

Postcondition:
None

Note:
When pLnModExpCb is non null, an asynchronous callback of type CpaCyLnModExpCbFunc is generated in response to this function call. Any errors generated during processing are reported in the structure returned in the callback.

See also:
CpaCyLnModExpOpData, **CpaCyGenFlatBufCbFunc**

CpaStatus cpaCyLnModInv (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyGenFlatBufCbFunc**	*pLnModInvCb*,
	void *	*pCallbackTag*,
	const **CpaCyLnModInvOpData** *	*pLnModInvOpData*,
	CpaFlatBuffer *	*pResult*
)		

File: cpa_cy_ln.h

Perform modular inversion operation.

This function performs modular inversion. It computes the following result based on the inputs:

result = (1/A) mod B.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pLnModInvCb*	Pointer to callback function to be invoked when the operation is complete.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pLnModInvOpData*	Structure containing all the data needed to perform the LN modular inversion operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pResult*	Pointer to a flat buffer containing a pointer to memory allocated by the client into which the result will be written. The size of the memory required MUST be larger than or equal to the size required to store the modulus. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.

CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized.

Postcondition:
None

Note:
When pLnModInvCb is non null, an asynchronous callback of type CpaCyLnModInvCbFunc is generated in response to this function call. Any errors generated during processing are reported in the structure returned in the callback.

See also:
CpaCyLnModInvOpData, **CpaCyGenFlatBufCbFunc**

CpaStatus CPA_DEPRECATED cpaCyLnStatsQuery (const **CpaInstanceHandle** *instanceHandle*, struct **_CpaCyLnStats** * *pLnStats*)

File: cpa_cy_ln.h

Query statistics for large number operations

Deprecated:
As of v1.3 of the Crypto API, this function has been deprecated, replaced by **cpaCyLnStatsQuery64()**.

This function will query a specific instance handle for large number statistics. The user MUST allocate the CpaCyLnStats structure and pass the reference to that structure into this function call. This function writes the statistic results into the passed in CpaCyLnStats structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[out]	*pLnStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Acceleration Services unit has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
CpaCyLnStats

CpaStatus cpaCyLnStatsQuery64 (const **CpaInstanceHandle** *instanceHandle*, **CpaCyLnStats64** * *pLnStats*)

File: cpa_cy_ln.h

Query statistics (64-bit version) for large number operations

This function will query a specific instance handle for the 64-bit version of the large number statistics. The user MUST allocate the CpaCyLnStats64 structure and pass the reference to that structure into this function call. This function writes the statistic results into the passed in CpaCyLnStats64 structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[out]	*pLnStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

Acceleration Services unit has been initialized.

Postcondition:

None

Note:

This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:

CpaCyLnStats

18 Prime Number Test API

[Cryptographic API]

Collaboration diagram for Prime Number Test API:

18.1 Detailed Description

File: cpa_cy_prime.h

These functions specify the API for the prime number test operations.

For prime number generation, this API SHOULD be used in conjunction with the Deterministic Random Bit Generation API (**Deterministic Random Bit Generation API**).

Note:

Large numbers are represented on the QuickAssist API as described in the Large Number API (**Cryptographic Large Number API**).

In addition, the bit length of large numbers passed to the API MUST NOT exceed 576 bits for Elliptic Curve operations.

18.2 Data Structures

- struct **_CpaCyPrimeTestOpData**
- struct **_CpaCyPrimeStats**
- struct **_CpaCyPrimeStats64**

18.3 Typedefs

- typedef **_CpaCyPrimeTestOpData CpaCyPrimeTestOpData**
- typedef **_CpaCyPrimeStats CPA_DEPRECATED**
- typedef **_CpaCyPrimeStats64 CpaCyPrimeStats64**
- typedef void(* **CpaCyPrimeTestCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** testPassed)

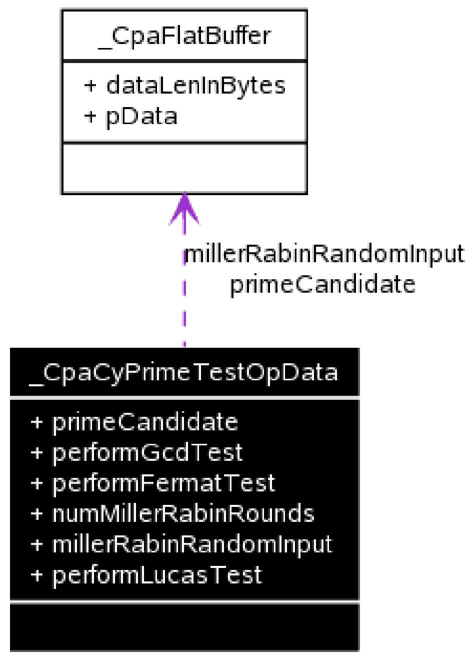
18.4 Functions

- **CpaStatus cpaCyPrimeTest** (const **CpaInstanceHandle** instanceHandle, const **CpaCyPrimeTestCbFunc** pCb, void *pCallbackTag, const **CpaCyPrimeTestOpData** *pOpData, **CpaBoolean** *pTestPassed)

18.5 Data Structure Documentation

18.5.1 _CpaCyPrimeTestOpData Struct Reference

Collaboration diagram for _CpaCyPrimeTestOpData:



18.5.1.1 Detailed Description

File: cpa_cy_prime.h

Prime Test Operation Data.

This structure contains the operation data for the cpaCyPrimeTest function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

All values in this structure are required to be in Most Significant Byte first order, e.g. primeCandidate.pData[0] = MSB.

All numbers MUST be stored in big-endian order.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyPrimeTest function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyPrimeTest()

18.5.1.2 Data Fields

- **CpaFlatBuffer primeCandidate**
- **CpaBoolean performGcdTest**
- **CpaBoolean performFermatTest**

- **Cpa32U numMillerRabinRounds**
- **CpaFlatBuffer millerRabinRandomInput**
- **CpaBoolean performLucasTest**

18.5.1.3 Field Documentation

CpaFlatBuffer _CpaCyPrimeTestOpData::primeCandidate

The prime number candidate to test

CpaBoolean _CpaCyPrimeTestOpData::performGcdTest

A value of CPA_TRUE means perform a GCD Primality Test

CpaBoolean _CpaCyPrimeTestOpData::performFermatTest

A value of CPA_TRUE means perform a Fermat Primality Test

Cpa32U _CpaCyPrimeTestOpData::numMillerRabinRounds

Number of Miller Rabin Primality Test rounds. Set to 0 to perform zero Miller Rabin tests. The maximum number of rounds supported is 50.

CpaFlatBuffer _CpaCyPrimeTestOpData::millerRabinRandomInput

Flat buffer containing a pointer to an array of n random numbers for Miller Rabin Primality Tests. The size of the buffer MUST be

n * (MAX(64,x))

where:

- n is the requested number of rounds.
- x is the minimum number of bytes required to represent the prime candidate, i.e. x = ceiling((ceiling(log2(p)))/8).

Each random number MUST be greater than 1 and less than the prime candidate - 1, with leading zeroes as necessary.

CpaBoolean _CpaCyPrimeTestOpData::performLucasTest

An CPA_TRUE value means perform a Lucas Primality Test

18.5.2 _CpaCyPrimeStats Struct Reference

18.5.2.1 Detailed Description

File: cpa_cy_prime.h

Prime Number Test Statistics.

Deprecated:

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyPrimeStats64**.

This structure contains statistics on the prime number test operations. Statistics are set to zero when the component is initialized, and are collected per instance.

18.5.2.2 Data Fields

- **Cpa32U numPrimeTestRequests**
- **Cpa32U numPrimeTestRequestErrors**
- **Cpa32U numPrimeTestCompleted**
- **Cpa32U numPrimeTestCompletedErrors**
- **Cpa32U numPrimeTestFailures**

18.5.2.3 Field Documentation

Cpa32U _CpaCyPrimeStats::numPrimeTestRequests

Total number of successful prime number test requests.

Cpa32U _CpaCyPrimeStats::numPrimeTestRequestErrors

Total number of prime number test requests that had an error and could not be processed.

Cpa32U _CpaCyPrimeStats::numPrimeTestCompleted

Total number of prime number test operations that completed successfully.

Cpa32U _CpaCyPrimeStats::numPrimeTestCompletedErrors

Total number of prime number test operations that could not be completed successfully due to errors.

Cpa32U _CpaCyPrimeStats::numPrimeTestFailures

Total number of prime number test operations that executed successfully but the outcome of the test was that the number was not prime.

18.5.3 _CpaCyPrimeStats64 Struct Reference

18.5.3.1 Detailed Description

File: cpa_cy_prime.h

Prime Number Test Statistics (64-bit version).

This structure contains a 64-bit version of the statistics on the prime number test operations. Statistics are set to zero when the component is initialized, and are collected per instance.

18.5.3.2 Data Fields

- **Cpa64U numPrimeTestRequests**
- **Cpa64U numPrimeTestRequestErrors**
- **Cpa64U numPrimeTestCompleted**
- **Cpa64U numPrimeTestCompletedErrors**
- **Cpa64U numPrimeTestFailures**

18.5.3.3 Field Documentation

Cpa64U _CpaCyPrimeStats64::numPrimeTestRequests

Total number of successful prime number test requests.

Cpa64U _CpaCyPrimeStats64::numPrimeTestRequestErrors

Total number of prime number test requests that had an error and could not be processed.

Cpa64U _CpaCyPrimeStats64::numPrimeTestCompleted

Total number of prime number test operations that completed successfully.

Cpa64U _CpaCyPrimeStats64::numPrimeTestCompletedErrors

Total number of prime number test operations that could not be completed successfully due to errors.

Cpa64U _CpaCyPrimeStats64::numPrimeTestFailures

Total number of prime number test operations that executed successfully but the outcome of the test was that the number was not prime.

18.6 Typedef Documentation

typedef struct **_CpaCyPrimeTestOpData CpaCyPrimeTestOpData**

File: cpa_cy_prime.h

Prime Test Operation Data.

This structure contains the operation data for the cpaCyPrimeTest function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

All values in this structure are required to be in Most Significant Byte first order, e.g. primeCandidate.pData[0] = MSB.

All numbers MUST be stored in big-endian order.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyPrimeTest function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyPrimeTest()

typedef struct **_CpaCyPrimeStats CPA_DEPRECATED**

File: cpa_cy_prime.h

Prime Number Test Statistics.

Deprecated:

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by **CpaCyPrimeStats64**.

This structure contains statistics on the prime number test operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef struct **_CpaCyPrimeStats64 CpaCyPrimeStats64**

File: cpa_cy_prime.h

Prime Number Test Statistics (64-bit version).

This structure contains a 64-bit version of the statistics on the prime number test operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef void(* **CpaCyPrimeTestCbFunc**)(void *pCallbackTag, **CpaStatus** status, void *pOpData, **CpaBoolean** testPassed)

File: cpa_cy_prime.h

Definition of callback function invoked for cpaCyPrimeTest requests.

This is the prototype for the cpaCyPrimeTest callback function.

Context:
This callback function can be executed in a context that DOES NOT permit sleeping to occur.

Assumptions:
None

Side-Effects:
None

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*pCallbackTag*	User-supplied value to help identify request.
`[in]`	*status*	Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.
`[in]`	*pOpData*	Opaque pointer to the Operation data pointer supplied in request.
`[in]`	*testPassed*	A value of CPA_TRUE means the prime candidate is probably prime.

Return values:
None

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
cpaCyPrimeTest()

18.7 Function Documentation

CpaStatus cpaCyPrimeTest (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyPrimeTestCbFunc**	*pCb*,
	void *	*pCallbackTag*,
	const **CpaCyPrimeTestOpData** *	*pOpData*,
	CpaBoolean *	*pTestPassed*
)		

File: cpa_cy_prime.h

Prime Number Test Function.

This function will test probabilistically if a number is prime. Refer to ANSI X9.80 2005 for details. The primality result will be returned in the asynchronous callback.

The following combination of GCD, Fermat, Miller-Rabin, and Lucas testing is supported: (up to 1x GCD) + (up to 1x Fermat) + (up to 50x Miller-Rabin rounds) + (up to 1x Lucas) For example: (1x GCD) + (25x Miller-Rabin) + (1x Lucas); (1x GCD) + (1x Fermat); (50x Miller-rabin);

Tests are always performed in order of increasing complexity, for example GCD first, then Fermat, then Miller-Rabin, and finally Lucas.

For all of the primality tests, the following prime number "sizes" (length in bits) are supported: all sizes up to and including 512 bits, as well as sizes 768, 1024, 1536, 2048, 3072 and 4096.

Candidate prime numbers MUST match these sizes accordingly, with leading zeroes present where necessary.

When this prime number test is used in conjunction with combined Miller-Rabin and Lucas tests, it may be used as a means of performing a self test operation on the random data generator.

A response status of ok (pass == CPA_TRUE) means all requested primality tests passed, and the prime candidate is probably prime (the exact probability depends on the primality tests requested). A response status of not ok (pass == CPA_FALSE) means one of the requested primality tests failed (the prime candidate has been found to be composite).

Context:
: When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
: None

Side-Effects:
: None

Blocking:
: Yes when configured to operate in synchronous mode.

Reentrant:
: No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pTestPassed*	A value of CPA_TRUE means the prime candidate is probably prime.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pCb is non-NULL an asynchronous callback of type CpaCyPrimeTestCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyPrimeTestOpData, **CpaCyPrimeTestCbFunc**

19 Deterministic Random Bit Generation API

[Cryptographic API]

Collaboration diagram for Deterministic Random Bit Generation API:

19.1 Detailed Description

File: cpa_cy_drbg.h

These functions specify the API for a Deterministic Random Bit Generation (DRBG), compliant with NIST SP 800-90, March 2007, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)".

The functions **cpaCyDrbgInitSession**, **cpaCyDrbgGen**, **cpaCyDrbgReseed** and **cpaCyDrbgRemoveSession** are used to instantiate, generate, reseed and uninstantiate a DRBG mechanism.

Note:

These functions supersede the random number generation functions in API group **Random Bit/Number Generation API**, which are now deprecated.

19.2 Data Structures

- struct **_CpaCyDrbgSessionSetupData**
- struct **_CpaCyDrbgGenOpData**
- struct **_CpaCyDrbgReseedOpData**
- struct **_CpaCyDrbgStats64**

19.3 Typedefs

- typedef enum **_CpaCyDrbgSecStrength CpaCyDrbgSecStrength**
- typedef **_CpaCyDrbgSessionSetupData CpaCyDrbgSessionSetupData**
- typedef void * **CpaCyDrbgSessionHandle**
- typedef **_CpaCyDrbgGenOpData CpaCyDrbgGenOpData**
- typedef **_CpaCyDrbgReseedOpData CpaCyDrbgReseedOpData**
- typedef **_CpaCyDrbgStats64 CpaCyDrbgStats64**

19.4 Enumerations

- enum **_CpaCyDrbgSecStrength** {
  CPA_CY_RBG_SEC_STRENGTH_112,
  CPA_CY_RBG_SEC_STRENGTH_128,
  CPA_CY_RBG_SEC_STRENGTH_192,
  CPA_CY_RBG_SEC_STRENGTH_256
  }

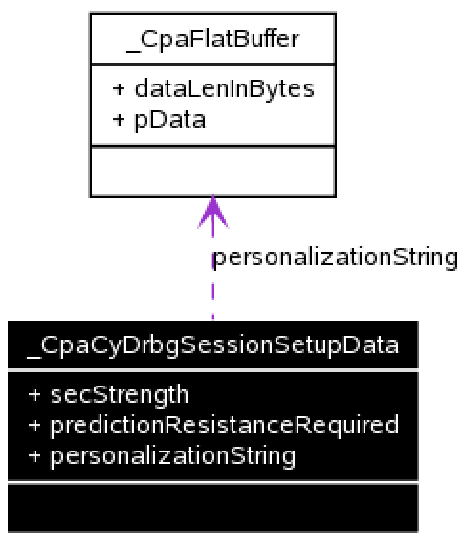
19.5 Functions

- **CpaStatus cpaCyDrbgSessionGetSize** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDrbgSessionSetupData** *pSetupData, **Cpa32U** *pSize)
- **CpaStatus cpaCyDrbgInitSession** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pGenCb, const **CpaCyGenericCbFunc** pReseedCb, const **CpaCyDrbgSessionSetupData** *pSetupData, **CpaCyDrbgSessionHandle** sessionHandle, **Cpa32U** *pSeedLen)
- **CpaStatus cpaCyDrbgReseed** (const **CpaInstanceHandle** instanceHandle, void *pCallbackTag, **CpaCyDrbgReseedOpData** *pOpData)
- **CpaStatus cpaCyDrbgGen** (const **CpaInstanceHandle** instanceHandle, void *pCallbackTag, **CpaCyDrbgGenOpData** *pOpData, **CpaFlatBuffer** *pPseudoRandomBits)
- **CpaStatus cpaCyDrbgRemoveSession** (const **CpaInstanceHandle** instanceHandle, **CpaCyDrbgSessionHandle** sessionHandle)
- **CpaStatus cpaCyDrbgQueryStats64** (const **CpaInstanceHandle** instanceHandle, **CpaCyDrbgStats64** *pStats)

19.6 Data Structure Documentation

19.6.1 _CpaCyDrbgSessionSetupData Struct Reference

Collaboration diagram for _CpaCyDrbgSessionSetupData:



19.6.1.1 Detailed Description

File: cpa_cy_drbg.h

DRBG Session (Instance) Setup Data

This structure contains data relating to instantiation of a DRBG session, or instance.

19.6.1.2 Data Fields

- **CpaCyDrbgSecStrength secStrength**
- **CpaBoolean predictionResistanceRequired**
- **CpaFlatBuffer personalizationString**

19.6.1.3 Field Documentation

CpaCyDrbgSecStrength _CpaCyDrbgSessionSetupData::secStrength

Requested security strength

CpaBoolean _CpaCyDrbgSessionSetupData::predictionResistanceRequired

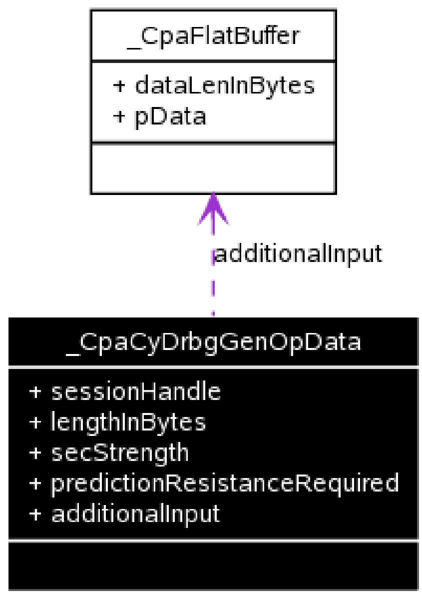
Prediction resistance flag. Indicates whether or not prediction resistance may be required by the consuming application during one or more requests for pseudorandom bits.

CpaFlatBuffer _CpaCyDrbgSessionSetupData::personalizationString

Personalization string. String that should be used to derive the seed.

19.6.2 _CpaCyDrbgGenOpData Struct Reference

Collaboration diagram for _CpaCyDrbgGenOpData:



19.6.2.1 Detailed Description

File: cpa_cy_drbg.h

DRBG Data Generation Operation Data

This structure contains data relating to generation of random bits using a DRBG.

See also:
: **cpaCyDrbgGen()**

Note:
: If the client modifies or frees the memory referenced in this structure after it has been submitted to the **cpaCyDrbgGen()** function, and before it has been returned in the callback, undefined behavior will result.

19.6.2.2 Data Fields

- **CpaCyDrbgSessionHandle sessionHandle**
- **Cpa32U lengthInBytes**
- **CpaCyDrbgSecStrength secStrength**
- **CpaBoolean predictionResistanceRequired**
- **CpaFlatBuffer additionalInput**

19.6.2.3 Field Documentation

CpaCyDrbgSessionHandle _CpaCyDrbgGenOpData::sessionHandle

Session handle, also known as the state handle or instance handle

Cpa32U _CpaCyDrbgGenOpData::lengthInBytes

Requested number of bytes to be generated

CpaCyDrbgSecStrength _CpaCyDrbgGenOpData::secStrength

Requested security strength

CpaBoolean _CpaCyDrbgGenOpData::predictionResistanceRequired

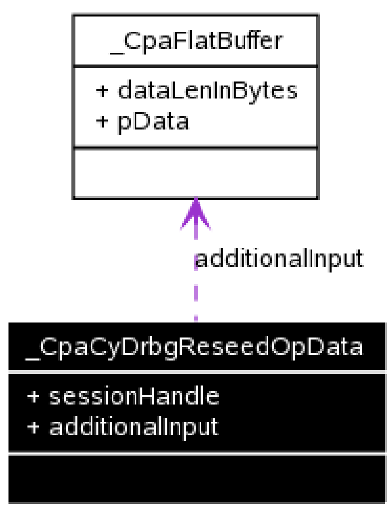
Requested prediction resistance flag. Indicates whether or not prediction resistance is to be provided prior to the generation of the requested pseudorandom bits to be generated.

CpaFlatBuffer _CpaCyDrbgGenOpData::additionalInput

Additional input

19.6.3 _CpaCyDrbgReseedOpData Struct Reference

Collaboration diagram for _CpaCyDrbgReseedOpData:



19.6.3.1 Detailed Description

File: cpa_cy_drbg.h

DRBG Reseed Operation Data

19.6.3 _CpaCyDrbgReseedOpData Struct Reference

This structure contains data relating to reseeding a DRBG session, or instance.

See also:
cpaCyDrbgReseed()

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the **cpaCyDrbgReseed()** function, and before it has been returned in the callback, undefined behavior will result.

19.6.3.2 Data Fields

- **CpaCyDrbgSessionHandle sessionHandle**
- **CpaFlatBuffer additionalInput**

19.6.3.3 Field Documentation

CpaCyDrbgSessionHandle _CpaCyDrbgReseedOpData::sessionHandle

Session handle, also known as a state handle or instance handle.

CpaFlatBuffer _CpaCyDrbgReseedOpData::additionalInput

An "optional" input to the reseeding. The length should be less than or equal to the seed length, which is returned by the function **cpaCyDrbgInitSession()**. A length of 0 can be specified to indicate no additional input.

19.6.4 _CpaCyDrbgStats64 Struct Reference

19.6.4.1 Detailed Description

File: cpa_cy_drbg.h

DRBG Statistics

This structure contains statistics (counters) related to the random bit generation API.

See also:
CpaCyDrbgQueryStats64()

19.6.4.2 Data Fields

- **Cpa64U numSessionsInitialized**
- **Cpa64U numSessionsRemoved**
- **Cpa64U numSessionErrors**
- **Cpa64U numGenRequests**
- **Cpa64U numGenRequestErrors**
- **Cpa64U numGenCompleted**
- **Cpa64U numGenCompletedErrors**
- **Cpa64U numReseedRequests**
- **Cpa64U numReseedRequestErrors**
- **Cpa64U numReseedCompleted**
- **Cpa64U numReseedCompletedErrors**

19.6.4.3 Field Documentation

Cpa64U _CpaCyDrbgStats64::numSessionsInitialized

Number of session initialized

Cpa64U _CpaCyDrbgStats64::numSessionsRemoved

Number of sessions removed

Cpa64U _CpaCyDrbgStats64::numSessionErrors

Total number of errors returned when initializing and removing sessions

Cpa64U _CpaCyDrbgStats64::numGenRequests

Number of successful calls to **cpaCyDrbgGen**.

Cpa64U _CpaCyDrbgStats64::numGenRequestErrors

Number of calls to **cpaCyDrbgGen** that returned an error and could not be processed.

Cpa64U _CpaCyDrbgStats64::numGenCompleted

Number of calls to **cpaCyDrbgGen** that completed successfully.

Cpa64U _CpaCyDrbgStats64::numGenCompletedErrors

Number of calls to **cpaCyDrbgGen** that completed with an error status.

Cpa64U _CpaCyDrbgStats64::numReseedRequests

Number of successful calls to **cpaCyDrbgReseed**.

Note that this does NOT include implicit reseeds due to calls to **cpaCyDrbgGen** with prediction resistance, or due to seed lifetime expiry.

Cpa64U _CpaCyDrbgStats64::numReseedRequestErrors

Number of calls to **cpaCyDrbgReseed** that returned an error and could not be processed.

Cpa64U _CpaCyDrbgStats64::numReseedCompleted

Number of calls to **cpaCyDrbgReseed** that completed successfully.

Cpa64U _CpaCyDrbgStats64::numReseedCompletedErrors

Number of calls to **cpaCyDrbgReseed** that completed with an error status.

19.7 Typedef Documentation

typedef enum **_CpaCyDrbgSecStrength CpaCyDrbgSecStrength**

File: cpa_cy_drbg.h

Security Strength

This enum defines the security strength. NIST SP 800-90 defines security strength as "A number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system; a security strength is specified in bits and is a specific value from the set

(112, 128, 192, 256) for this Recommendation. The amount of work needed is 2^(security_strength)."

typedef struct **_CpaCyDrbgSessionSetupData CpaCyDrbgSessionSetupData**

File: cpa_cy_drbg.h

DRBG Session (Instance) Setup Data

This structure contains data relating to instantiation of a DRBG session, or instance.

typedef void* **CpaCyDrbgSessionHandle**

File: cpa_cy_drbg.h

Handle to a DRBG session (or instance).

This is what NIST SP 800-90 refers to as the "state_handle". That document also refers to the process of creating such a handle as "instantiation", or instance creation. On this API, we use the term "session" to refer to such an instance, to avoid confusion with the crypto instance handle, and for consistency with the similar concept of sessions in symmetric crypto (see **Symmetric Cipher and Hash Cryptographic API**) and elsewhere on the API.

Note that there can be multiple sessions, or DRBG instances, created within a single instance of a CpaInstanceHandle.

Note:

The memory for this handle is allocated by the client. The size of the memory that the client needs to allocate is determined by a call to the **cpaCyDrbgSessionGetSize** function. The session memory is initialized with a call to the **cpaCyDrbgInitSession** function. This memory MUST not be freed until a call to **cpaCyDrbgRemoveSession** has completed successfully.

typedef struct **_CpaCyDrbgGenOpData CpaCyDrbgGenOpData**

File: cpa_cy_drbg.h

DRBG Data Generation Operation Data

This structure contains data relating to generation of random bits using a DRBG.

See also:

cpaCyDrbgGen()

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the **cpaCyDrbgGen()** function, and before it has been returned in the callback, undefined behavior will result.

typedef struct **_CpaCyDrbgReseedOpData CpaCyDrbgReseedOpData**

File: cpa_cy_drbg.h

DRBG Reseed Operation Data

This structure contains data relating to reseeding a DRBG session, or instance.

See also:
cpaCyDrbgReseed()

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the **cpaCyDrbgReseed()** function, and before it has been returned in the callback, undefined behavior will result.

typedef struct **_CpaCyDrbgStats64 CpaCyDrbgStats64**

File: cpa_cy_drbg.h

DRBG Statistics

This structure contains statistics (counters) related to the random bit generation API.

See also:
CpaCyDrbgQueryStats64()

19.8 Enumeration Type Documentation

enum **_CpaCyDrbgSecStrength**

File: cpa_cy_drbg.h

Security Strength

This enum defines the security strength. NIST SP 800-90 defines security strength as "A number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system; a security strength is specified in bits and is a specific value from the set (112, 128, 192, 256) for this Recommendation. The amount of work needed is 2^(security_strength)."

19.9 Function Documentation

CpaStatus cpaCyDrbgSessionGetSize (const **CpaInstanceHandle** *instanceHandle*,
const **CpaCyDrbgSessionSetupData** * *pSetupData*,
Cpa32U * *pSize*
)

File: cpa_cy_drbg.h

Returns the size (in bytes) of a DRBG session handle.

This function is used by the client to determine the size of the memory it must allocate in order to store the DRBG session. This MUST be called before the client allocates the memory for the session and before the client calls the **cpaCyDrbgInitSession** function.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:

None

Side-Effects:
None

Blocking:
No.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pSetupData*	Pointer to session setup data which contains parameters which are static for a given DRBG session, such as security strength, etc.
[out]	*pSize*	The amount of memory in bytes required to hold the session.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via the **cpaCyStartInstance** function.

Postcondition:
None

CpaStatus cpaCyDrbgInitSession	(	const **CpaInstanceHandle**	*instanceHandle*,
		const **CpaCyGenFlatBufCbFunc**	*pGenCb*,
		const **CpaCyGenericCbFunc**	*pReseedCb*,
		const **CpaCyDrbgSessionSetupData** *	*pSetupData*,
		CpaCyDrbgSessionHandle	*sessionHandle*,
		Cpa32U *	*pSeedLen*
	)		

File: cpa_cy_drbg.h

Instantiates and seeds a DRBG session, or instance.

This function is used by the client to initialize a DRBG session, or instance.

Note:
On some implementations, the client may have to register an entropy source, nonce source, and/or a function which specifies whether a derivation function is required. See the Programmer's Guide for your implementation for more details.

Context:

This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
No.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pGenCb*	Pointer to callback function to be registered. This is the function that will be called back to indicate completion of the asynchronous **cpaCyDrbgGen** function. Set this field to NULL if this function is to operate in a synchronous manner.
[in]	*pReseedCb*	Pointer to callback function to be registered. This is the function that will be called back to indicate completion of the asynchronous **cpaCyDrbgReseed** function. Set this field to NULL if this function is to operate in a synchronous manner.
[in]	*pSetupData*	Pointer to setup data.
[out]	*sessionHandle*	Pointer to the memory allocated by the client to store the instance handle. This will be initialized with this function. This handle needs to be passed to subsequent processing calls.
[out]	*pSeedLen*	Seed length for the supported DRBG mechanism and security strength. The value of this is dependent on the DRBG mechanism implemented by the instance, which is implementation-dependent. This seed length may be used by the client when reseeding.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via the **cpaCyStartInstance** function.

Postcondition:
None

CpaStatus cpaCyDrbgReseed	(	const **CpaInstanceHandle**	*instanceHandle*,
		void *	*pCallbackTag*,
		CpaCyDrbgReseedOpData *	*pOpData*
	)		

File: cpa_cy_drbg.h

Reseeds a DRBG session, or instance.

Reseeding inserts additional entropy into the generation of pseudorandom bits.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via the **cpaCyStartInstance** function.

Postcondition:
None

CpaStatus cpaCyDrbgGen (	const **CpaInstanceHandle**	*instanceHandle*,
	void *	*pCallbackTag*,
	CpaCyDrbgGenOpData *	*pOpData*,
	CpaFlatBuffer *	*pPseudoRandomBits*
)		

File: cpa_cy_drbg.h

Generates pseudorandom bits.

This function is used to request the generation of random bits. The generated data and the length of the data will be returned to the caller in an asynchronous callback function.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pPseudoRandomBits*	Pointer to the memory allocated by the client where the random data will be written to. For optimal performance, the data pointed to SHOULD be 8-byte aligned. There is no endianness associated with the random data. On invocation the callback function will contain this parameter in its pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources. One reason may be for an entropy test failing.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via the **cpaCyStartInstance** function. The DRBG session, or instance, has been initialized via the **cpaCyDrbgInitSession** function.

Postcondition:
None

CpaStatus cpaCyDrbgRemoveSession (const **CpaInstanceHandle** *instanceHandle*, **CpaCyDrbgSessionHandle** *sessionHandle*)

File: cpa_cy_drbg.h

Removes a previously instantiated DRBG session, or instance.

This function will remove a previously initialized DRBG session, or instance, and the installed callback handler function. Removal will fail if outstanding calls still exist for the initialized session. In this case, the client needs to retry the remove function at a later time. The memory for the session handle MUST not be freed until this call has completed successfully.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
No.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*sessionHandle*	DRBG session handle to be removed.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized via the **cpaCyStartInstance** function. The DRBG session, or instance, has been initialized via the **cpaCyDrbgInitSession** function.

Postcondition:
None

CpaStatus cpaCyDrbgQueryStats64 (	const **CpaInstanceHandle**	*instanceHandle*,
	CpaCyDrbgStats64 *	*pStats*
)		

File: cpa_cy_drbg.h

Returns statistics specific to a session, or instance, of the RBG API.

This function will query a specific session for RBG statistics. The user MUST allocate the CpaCyDrbgStats64 structure and pass the reference to that into this function call. This function writes the statistic results into the passed in CpaCyDrbgStats64 structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[out]`	*pStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
Component has been initialized.

Postcondition:
None

20 Non-Deterministic Random Bit Generation API

[Cryptographic API]

Collaboration diagram for Non-Deterministic Random Bit Generation API:

20.1 Detailed Description

File: cpa_cy_nrbg.h

These functions specify the API for Non-Deterministic Random Bit Generation (NRBG). This is used to provide entropy to a Deterministic RBG (DRBG).

Note:

These functions supersede the random number generation functions in API group **Random Bit/Number Generation API**, which are now deprecated.

20.2 Data Structures

- struct **_CpaCyNrbgOpData**

20.3 Typedefs

- typedef **_CpaCyNrbgOpData CpaCyNrbgOpData**

20.4 Functions

- **CpaStatus cpaCyNrbgGetEntropy** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pCb, void *pCallbackTag, const **CpaCyNrbgOpData** *pOpData, **CpaFlatBuffer** *pEntropy)

20.5 Data Structure Documentation

20.5.1 _CpaCyNrbgOpData Struct Reference

20.5.1.1 Detailed Description

File: cpa_cy_nrbg.h

NRBG Get Entropy Operation Data

This structure contains data relating to generation of entropy using an NRBG.

See also:

cpaCyNrbgGetEntropy()

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the **cpaCyNrbgGetEntropy()** function, and before it has been returned in the callback, undefined behavior will result.

20.5.1.2 Data Fields

- **Cpa32U lengthInBytes**

20.5.1.3 Field Documentation

Cpa32U _CpaCyNrbgOpData::lengthInBytes

Requested number of bytes to be generated. On calls to **cpaCyNrbgGetEntropy**, this value must be greater than zero (>0).

20.6 Typedef Documentation

typedef struct **_CpaCyNrbgOpData CpaCyNrbgOpData**

File: cpa_cy_nrbg.h

NRBG Get Entropy Operation Data

This structure contains data relating to generation of entropy using an NRBG.

See also:
cpaCyNrbgGetEntropy()

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the **cpaCyNrbgGetEntropy()** function, and before it has been returned in the callback, undefined behavior will result.

20.7 Function Documentation

CpaStatus cpaCyNrbgGetEntropy (const **CpaInstanceHandle** *instanceHandle*,
const **CpaCyGenFlatBufCbFunc** *pCb*,
void * *pCallbackTag*,
const **CpaCyNrbgOpData** * *pOpData*,
CpaFlatBuffer * *pEntropy*
)

File: cpa_cy_rbg.h

Gets entropy from the NRBG.

This function returns a string of bits of specified length.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCb*	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pEntropy*	Pointer to memory allocated by the client to which the entropy will be written. For optimal performance, the data pointed to SHOULD be 8-byte aligned. There is no endianness associated with the entropy. On invocation the callback function will contain this parameter in its pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via the **cpaCyStartInstance** function.

Postcondition:
None

Note:
When pCb is non-NULL an asynchronous callback of type **CpaCyGenFlatBufCbFunc** is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. For optimal performance, data pointers SHOULD be 8-byte aligned.

21 Random Bit/Number Generation API

[Cryptographic API]

Collaboration diagram for Random Bit/Number Generation API:

21.1 Detailed Description

File: cpa_cy_rand.h

Deprecated:
As of v1.3 of the API, this entire API group has been deprecated, replaced by API groups **Deterministic Random Bit Generation API** and **Non-Deterministic Random Bit Generation API** .

These functions specify the API for the Cryptographic Random Bit and Random number generation.

21.2 Data Structures

- struct **_CpaCyRandStats**
- struct **_CpaCyRandGenOpData**
- struct **_CpaCyRandSeedOpData**

21.3 Defines

- #define **CPA_CY_RAND_SEED_LEN_IN_BYTES**

21.4 Typedefs

- typedef **_CpaCyRandStats CPA_DEPRECATED**
- typedef **_CpaCyRandGenOpData CPA_DEPRECATED**
- typedef **_CpaCyRandSeedOpData CPA_DEPRECATED**

21.5 Functions

- **CpaStatus CPA_DEPRECATED cpaCyRandGen** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pRandGenCb, void *pCallbackTag, const struct **_CpaCyRandGenOpData** *pRandGenOpData, **CpaFlatBuffer** *pRandData)
- **CpaStatus CPA_DEPRECATED cpaCyRandSeed** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenericCbFunc** pRandSeedCb, void *pCallbackTag, const struct **_CpaCyRandSeedOpData** *pSeedOpData)
- **CpaStatus CPA_DEPRECATED cpaCyRandQueryStats** (const **CpaInstanceHandle** instanceHandle, struct **_CpaCyRandStats** *pRandStats)

21.6 Data Structure Documentation

21.6.1 _CpaCyRandStats Struct Reference

21.6.1.1 Detailed Description

File: cpa_cy_rand.h

Random Data Generator Statistics.

Deprecated:

As of v1.3 of the API, replaced by **CpaCyDrbgStats64**.

This structure contains statistics on the random data generation operations. Statistics are set to zero when the component is initialized, and are collected per instance.

21.6.1.2 Data Fields

- **Cpa32U numRandNumRequests**
- **Cpa32U numRandNumRequestErrors**
- **Cpa32U numRandNumCompleted**
- **Cpa32U numRandNumCompletedErrors**
- **Cpa32U numRandBitRequests**
- **Cpa32U numRandBitRequestErrors**
- **Cpa32U numRandBitCompleted**
- **Cpa32U numRandBitCompletedErrors**
- **Cpa32U numNumSeedRequests**
- **Cpa32U numRandSeedCompleted**
- **Cpa32U numNumSeedErrors**

21.6.1.3 Field Documentation

Cpa32U _CpaCyRandStats::numRandNumRequests

Total number of successful random number generation requests.

Cpa32U _CpaCyRandStats::numRandNumRequestErrors

Total number of random number generation requests that had an error and could not be processed.

Cpa32U _CpaCyRandStats::numRandNumCompleted

Total number of random number operations that completed successfully.

Cpa32U _CpaCyRandStats::numRandNumCompletedErrors

Total number of random number operations that could not be completed successfully due to errors.

Cpa32U _CpaCyRandStats::numRandBitRequests

Total number of successful random bit generation requests.

Cpa32U _CpaCyRandStats::numRandBitRequestErrors

Total number of random bit generation requests that had an error and could not be processed.

Cpa32U _CpaCyRandStats::numRandBitCompleted

Total number of random bit operations that completed successfully.

Cpa32U _CpaCyRandStats::numRandBitCompletedErrors

Total number of random bit operations that could not be completed successfully due to errors.

Cpa32U _CpaCyRandStats::numNumSeedRequests

Total number of seed operations requests.

Cpa32U _CpaCyRandStats::numRandSeedCompleted

Total number of seed operations completed.

Cpa32U _CpaCyRandStats::numNumSeedErrors

Total number of seed operation errors.

21.6.2 _CpaCyRandGenOpData Struct Reference

21.6.2.1 Detailed Description

File: cpa_cy_rand.h

Random Bit/Number Generation Data.

Deprecated:

As of v1.3 of the API, replaced by **CpaCyDrbgGenOpData**.

This structure lists the different items that are required in the cpaCyRandGen function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyRandGen function, and before it has been returned in the callback, undefined behavior will result.

21.6.2.2 Data Fields

- **CpaBoolean generateBits**
- **Cpa32U lenInBytes**

21.6.2.3 Field Documentation

CpaBoolean _CpaCyRandGenOpData::generateBits

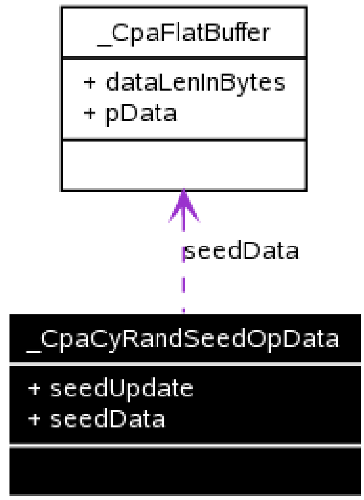
When set to CPA_TRUE then the cpaCyRandGen function will generate random bits which will comply with the ANSI X9.82 Part 1 specification. When set to CPA_FALSE random numbers will be produced from the random bits generated by the hardware. This will be spec compliant in terms of the probability of the random nature of the number returned.

Cpa32U _CpaCyRandGenOpData::lenInBytes

Specifies the length in bytes of the data returned. If the data returned is a random number, then it is implicit that the random number will fall into the following range: Expressed mathematically, the range is [2^(lenInBytes*8 - 1) to 2^(lenInBytes*8) - 1]. This is equivalent to "1000...0000" to "1111...1111" which requires (lenInBytes * 8) bits to represent. The maximum number of random bytes that can be requested is 65535 bytes.

21.6.3 _CpaCyRandSeedOpData Struct Reference

Collaboration diagram for _CpaCyRandSeedOpData:



21.6.3.1 Detailed Description

File: cpa_cy_rand.h

Random Generator Seed Data.

Deprecated:
As of v1.3 of the API, replaced by **CpaCyDrbgReseedOpData**.

This structure lists the different items that required in the cpaCyRandSeed function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyRandSeed function, and before it has been returned in the callback, undefined behavior will result.

21.6.3.2 Data Fields

- **CpaBoolean seedUpdate**
- **CpaFlatBuffer seedData**

21.6.3.3 Field Documentation

CpaBoolean _CpaCyRandSeedOpData::seedUpdate

When set to CPA_TRUE then the cpaCyRandSeed function will update (combine) the specified seed with the stored seed. When set to CPA_FALSE, the cpaCyRandSeed function will completely discard all existing entropy in the hardware and replace with the specified seed.

CpaFlatBuffer _CpaCyRandSeedOpData::seedData

Data for use in either seeding or performing a seed update. The data that is pointed to are random bits and as such do not have an endian order. For optimal performance the data SHOULD be 8-byte aligned. The length of the seed data is in bytes. This MUST currently be equal to CPA_CY_RAND_SEED_LEN_IN_BYTES.

21.7 Define Documentation

#define CPA_CY_RAND_SEED_LEN_IN_BYTES

File: cpa_cy_rand.h

Random Bit/Number Generator Seed Length

Defines the permitted seed length in bytes that may be used with the cpaCyRandSeed function.

See also:
cpaCyRandSeed

21.8 Typedef Documentation

typedef struct **_CpaCyRandStats CPA_DEPRECATED**

File: cpa_cy_rand.h

Random Data Generator Statistics.

Deprecated:
As of v1.3 of the API, replaced by **CpaCyDrbgStats64**.

This structure contains statistics on the random data generation operations. Statistics are set to zero when the component is initialized, and are collected per instance.

typedef struct **_CpaCyRandGenOpData CPA_DEPRECATED**

File: cpa_cy_rand.h

Random Bit/Number Generation Data.

Deprecated:
As of v1.3 of the API, replaced by **CpaCyDrbgGenOpData**.

This structure lists the different items that are required in the cpaCyRandGen function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyRandGen function, and before it has been returned in the callback, undefined behavior will result.

typedef struct **_CpaCyRandSeedOpData CPA_DEPRECATED**

File: cpa_cy_rand.h

Random Generator Seed Data.

Deprecated:
As of v1.3 of the API, replaced by **CpaCyDrbgReseedOpData**.

This structure lists the different items that required in the cpaCyRandSeed function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

Note:
If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyRandSeed function, and before it has been returned in the callback, undefined behavior will result.

21.9 Function Documentation

CpaStatus CPA_DEPRECATED cpaCyRandGen	(const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyGenFlatBufCbFunc**	*pRandGenCb*,
	void *	*pCallbackTag*,
	const struct **_CpaCyRandGenOpData** *	*pRandGenOpData*,
	CpaFlatBuffer *	*pRandData*
	)	

File: cpa_cy_rand.h

Random Bits or Number Generation Function.

Deprecated:
As of v1.3 of the API, replaced by **cpaCyDrbgGen()**.

This function is used to request the generation of random bits or a random number. The generated data and the length of the data will be returned to the caller in an asynchronous callback function. If random number generation is selected, the random bits generated by the hardware will be converted to a random number that is compliant to the ANSI X9.82 Part 1 specification.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:

Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pRandGenCb*	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in]	*pRandGenOpData*	Structure containing all the data needed to perform the random bit/number operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pRandData*	Pointer to the memory allocated by the client where the random data will be written to. For optimal performance, the data pointed to SHOULD be 8-byte aligned. There is no endianness associated with the random data. On invocation the callback function will contain this parameter in the pOut parameter.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources. One reason may be for an entropy test failing.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pRandGenCb is non-NULL an asynchronous callback of type CpaCyRandGenCbFunc is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. Entropy testing and reseeding are performed automatically by this function.

See also:
CpaCyGenFlatBufCbFunc, CpaCyRandGenOpData, **cpaCyRandSeed()**.

CpaStatus CPA_DEPRECATED
cpaCyRandSeed (const **CpaInstanceHandle** *instanceHandle*,
const **CpaCyGenericCbFunc** *pRandSeedCb*,
void * *pCallbackTag*,
const struct **_CpaCyRandSeedOpData** * *pSeedOpData*

)

File: cpa_cy_rand.h

Random Data Generator Seed Function.

Deprecated:
As of v1.3 of the API, replaced by **cpaCyDrbgReseed()**.

This function is used to either seed or perform a seed update on the random data generator. Replacing the seed with a user supplied seed value, or performing a seed update are completely optional operations. If seeding is specified, it has the effect or disregarding all existing entropy within the random data generator and replacing with the specified seed. If performing a seed update, then the specified seed is mixed into the stored seed. The seed length MUST be equal to CPA_CY_RAND_SEED_LEN_IN_BYTES.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pRandSeedCb*	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in]	*pSeedOpData*	Structure containing all the data needed to perform the random generator seed operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pRandSeedCn is non-NULL an asynchronous callback of type CpaCyRandSeedCbFunc is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. Entropy testing and reseeding are performed automatically by the cpaCyRandGen function.

See also:
CpaCyGenericCbFunc, CpaCyRandSeedOpData, **cpaCyRandGen()**

CpaStatus CPA_DEPRECATED cpaCyRandQueryStats (const **CpaInstanceHandle** *instanceHandle*, struct **_CpaCyRandStats** * *pRandStats*)

File: cpa_cy_rand.h

Query random number statistics specific to an instance.

Deprecated:
As of v1.3 of the API, replaced by **cpaCyDrbgQueryStats64()**.

This function will query a specific instance for random number statistics. The user MUST allocate the CpaCyRandStats structure and pass the reference to that into this function call. This function will write the statistic results into the passed in CpaCyRandStats structure.

Note: statistics returned by this function do not interrupt current data processing and as such can be slightly out of sync with operations that are in progress during the statistics retrieval process.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[out]`	*pRandStats*	Pointer to memory into which the statistics will be written.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

Component has been initialized.

Postcondition:

None

Note:

This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:

CpaCyRandStats

22 Intel(R) Key Protection Technology (KPT) Cryptographic API

[Cryptographic API]

Collaboration diagram for Intel(R) Key Protection Technology (KPT) Cryptographic API:

22.1 Detailed Description

File: cpa_cy_kpt.h

These functions specify the APIs for Key Protection Technology (KPT) Cryptographic services.

Note:

These functions implement the KPT Cryptographic API, In order to realize full KPT function, you need Intel(R) PTT (Platform Trust Technology) and Intel C62X PCH support, which provide 1. QuickAssist Technology 2. Trusted Platform Module (TPM2.0) 3. Secure communication channel between QAT and PTT

22.2 Data Structures

- struct **CpaCyKptWrappingFormat_t**
- struct **CpaCyKptRsaWpkSizeRep2_t**
- union **CpaCyKptWpkSize_t**
- struct **CpaCyKptUnwrapContext_t**
- struct **_CpaCyKptEcdsaSignRSOpData**

22.3 Defines

- #define **CPA_CY_KPT_MAX_IV_LENGTH**
- #define **CPA_CY_KPT_HMAC_LENGTH**
- #define **CPA_CY_KPT_CALLER_NONCE_LENGTH**
- #define **CPA_CY_KPT_DEVICE_NONCE_LENGTH**

22.4 Typedefs

- typedef **Cpa64U CpaCyKptHandle**
- typedef enum **CpaCyKptWrappingKeyType_t CpaCyKptWrappingKeyType**
- typedef enum **CpaCyKptHMACType_t CpaCyKptHMACType**
- typedef enum **CpaCyKptKeyManagementStatus_t CpaCyKptKeyManagementStatus**
- typedef enum **CpaCyKptKeySelectionFlags_t CpaCyKptKeySelectionFlags**
- typedef enum **CpaCyKptKeyAction_t CpaCyKptKeyAction**
- typedef **CpaCyKptWrappingFormat_t CpaCyKptWrappingFormat**
- typedef **CpaCyKptRsaWpkSizeRep2_t CpaCyKptRsaWpkSizeRep2**
- typedef **CpaCyKptWpkSize_t CpaCyKptWpkSize**
- typedef **CpaCyKptUnwrapContext_t CpaCyKptUnwrapContext**
- typedef **_CpaCyKptEcdsaSignRSOpData CpaCyKptEcdsaSignRSOpData**

22.5 Enumerations

- enum **CpaCyKptWrappingKeyType_t** {
  CPA_CY_KPT_WRAPPING_KEY_TYPE_AES128_GCM,
  CPA_CY_KPT_WRAPPING_KEY_TYPE_AES256_GCM,
  CPA_CY_KPT_WRAPPING_KEY_TYPE_AES128_CBC,
  CPA_CY_KPT_WRAPPING_KEY_TYPE_AES256_CBC
  }
- enum **CpaCyKptHMACType_t** {
  CPA_CY_KPT_HMAC_TYPE_NULL,
  CPA_CY_KPT_HMAC_TYPE_SHA1,
  CPA_CY_KPT_HMAC_TYPE_SHA224,
  CPA_CY_KPT_HMAC_TYPE_SHA256,
  CPA_CY_KPT_HMAC_TYPE_SHA384,
  CPA_CY_KPT_HMAC_TYPE_SHA512,
  CPA_CY_KPT_HMAC_TYPE_SHA3_224,
  CPA_CY_KPT_HMAC_TYPE_SHA3_256,
  CPA_CY_KPT_HMAC_TYPE_SHA3_384,
  CPA_CY_KPT_HMAC_TYPE_SHA3_512
  }
- enum **CpaCyKptKeyManagementStatus_t** {
  CPA_CY_KPT_SUCCESS,
  CPA_CY_KPT_REGISTER_HANDLE_FAIL_RETRY,
  CPA_CY_KPT_REGISTER_HANDLE_FAIL_DUPLICATE,
  CPA_CY_KPT_LOAD_KEYS_FAIL_INVALID_HANDLE,
  CPA_CY_KPT_REGISTER_HANDLE_FAIL_WKT_FULL,
  CPA_CY_KPT_WKT_ENTRY_NOT_FOUND,
  CPA_CY_KPT_REGISTER_HANDLE_FAIL_INSTANCE_QUOTA_EXCEEDED,
  CPA_CY_KPT_LOADKEYS_FAIL_CHECKSUM_ERROR,
  CPA_CY_KPT_LOADKEYS_FAIL_HANDLE_NOT_REGISTERED,
  CPA_CY_KPT_LOADKEYS_FAIL_POSSIBLE_DOS_ATTACK,
  CPA_CY_KPT_LOADKEYS_FAIL_INVALID_AC_SEND_HANDLE,
  CPA_CY_KPT_LOADKEYS_FAIL_INVALID_DATA_OBJ,
  CPA_CY_KPT_FAILED
  }
- enum **CpaCyKptKeySelectionFlags_t** {
  CPA_CY_KPT_SWK,
  CPA_CY_KPT_WPK,
  CPA_CY_KPT_OPAQUE_DATA,
  CPA_CY_KPT_HMAC_AUTH_PARAMS,
  CPA_CY_KPT_RN_SEED
  }
- enum **CpaCyKptKeyAction_t** {
  CPA_CY_KPT_NO_HMAC_AUTH_CHECK,
  CPA_CY_KPT_HMAC_AUTH_CHECK
  }

22.6 Functions

- **CpaStatus cpaCyKptRegisterKeyHandle** (**CpaInstanceHandle** instanceHandle, **CpaCyKptHandle** keyHandle, **CpaCyKptKeyManagementStatus** *pKptStatus)
- **CpaStatus cpaCyKptLoadKeys** (**CpaInstanceHandle** instanceHandle, **CpaCyKptHandle** keyHandle, **CpaCyKptWrappingFormat** *pKptWrappingFormat, **CpaCyKptKeySelectionFlags** keySelFlag, **CpaCyKptKeyAction** keyAction, **CpaFlatBuffer** *pOutputData, **CpaCyKptKeyManagementStatus** *pKptStatus)

- **CpaStatus cpaCyKptDeleteKey** (**CpaInstanceHandle** instanceHandle, **CpaCyKptHandle** keyHandle, **CpaCyKptKeyManagementStatus** *pkptstatus)
- **CpaStatus cpaCyKptRsaDecrypt** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pRsaDecryptCb, void *pCallbackTag, const **CpaCyRsaDecryptOpData** *pDecryptOpData, **CpaFlatBuffer** *pOutputData, **CpaFlatBuffer** *pKptUnwrapContext)
- **CpaStatus cpaCyKptEcdsaSignRS** (const **CpaInstanceHandle** instanceHandle, const **CpaCyEcdsaSignRSCbFunc** pCb, void *pCallbackTag, const **CpaCyKptEcdsaSignRSOpData** *pOpData, **CpaBoolean** *pSignStatus, **CpaFlatBuffer** *pR, **CpaFlatBuffer** *pS, **CpaFlatBuffer** *pKptUnwrapContext)
- **CpaStatus cpaCyKptDsaSignS** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaGenCbFunc** pCb, void *pCallbackTag, const **CpaCyDsaSSignOpData** *pOpData, **CpaBoolean** *pProtocolStatus, **CpaFlatBuffer** *pS, **CpaFlatBuffer** *pKptUnwrapContext)
- **CpaStatus cpaCyKptDsaSignRS** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaRSSignCbFunc** pCb, void *pCallbackTag, const **CpaCyDsaRSSignOpData** *pOpData, **CpaBoolean** *pProtocolStatus, **CpaFlatBuffer** *pR, **CpaFlatBuffer** *pS, **CpaFlatBuffer** *pKptUnwrapContext)

22.7 Data Structure Documentation

22.7.1 CpaCyKptWrappingFormat_t Struct Reference

22.7.1.1 Detailed Description

File: cpa_cy_kpt.h

KPT wrapping format structure.

This structure defines wrapping format which is used to wrap clear private keys using a symmetric wrapping key.Application sets these parameters through the cpaCyKptLoadKeys calls.

22.7.1.2 Data Fields

- **CpaCyKptWrappingKeyType wrappingAlgorithm**
- **Cpa8U iv** [CPA_CY_KPT_MAX_IV_LENGTH]
- **Cpa32U iterationCount**
- **CpaCyKptHMACType hmacType**

22.7.1.3 Field Documentation

CpaCyKptWrappingKeyType CpaCyKptWrappingFormat_t::wrappingAlgorithm

Symmetric wrapping algorithm

Cpa8U CpaCyKptWrappingFormat_t::iv[CPA_CY_KPT_MAX_IV_LENGTH]

Initialization Vector

Cpa32U CpaCyKptWrappingFormat_t::iterationCount

Iteration Count for Key Wrap Algorithms

CpaCyKptHMACType CpaCyKptWrappingFormat_t::hmacType

Hash algorithm used in WPK tag generation

22.7.2 CpaCyKptRsaWpkSizeRep2_t Struct Reference

22.7.2.1 Detailed Description

File: cpa_cy_kpt.h

RSA wrapped private key size structure For Representation 2.

This structure contains byte length of wrapped quintuple of p,q,dP,dQ and qInv which are required for the second representation of RSA private key. PKCS #1 V2.1 specification defines the second representation of the RSA private key,The quintuple of p, q, dP, dQ, and qInv are required for this representation.
CpaCyRsaPrivateKeyRep2

22.7.2.2 Data Fields

- **Cpa32U pLenInBytes**
- **Cpa32U qLenInBytes**
- **Cpa32U dpLenInBytes**
- **Cpa32U dqLenInBytes**
- **Cpa32U qinvLenInBytes**

22.7.2.3 Field Documentation

Cpa32U CpaCyKptRsaWpkSizeRep2_t::pLenInBytes

The byte length of wrapped prime p

Cpa32U CpaCyKptRsaWpkSizeRep2_t::qLenInBytes

The byte length of wrapped prime q

Cpa32U CpaCyKptRsaWpkSizeRep2_t::dpLenInBytes

The byte length of wrapped factor CRT exponent (dP)

Cpa32U CpaCyKptRsaWpkSizeRep2_t::dqLenInBytes

The byte length of wrapped factor CRT exponent (dQ)

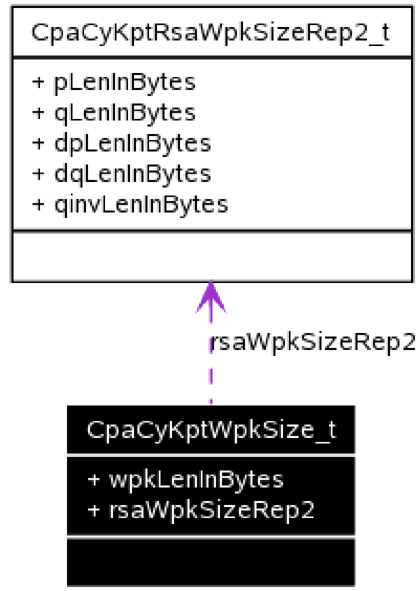
Cpa32U CpaCyKptRsaWpkSizeRep2_t::qinvLenInBytes

The byte length of wrapped coefficient (qInv)

22.7.3 CpaCyKptWpkSize_t Union Reference

Collaboration diagram for CpaCyKptWpkSize_t:

22.7.3 CpaCyKptWpkSize_t Union Reference



22.7.3.1 Detailed Description

File: cpa_cy_kpt.h

Wrapped private key size union.

A wrapped private key size union, either wrapped quintuple of RSA representation 2 private key,or byte length of wrapped ECC/RSA Rep1/DSA/ ECDSA private key.

22.7.3.2 Data Fields

- **Cpa32U wpkLenInBytes**
- **CpaCyKptRsaWpkSizeRep2 rsaWpkSizeRep2**

22.7.3.3 Field Documentation

Cpa32U CpaCyKptWpkSize_t::wpkLenInBytes

The byte length of wrapped private key for Rsa rep1,ECC,DSA and ECDSA case

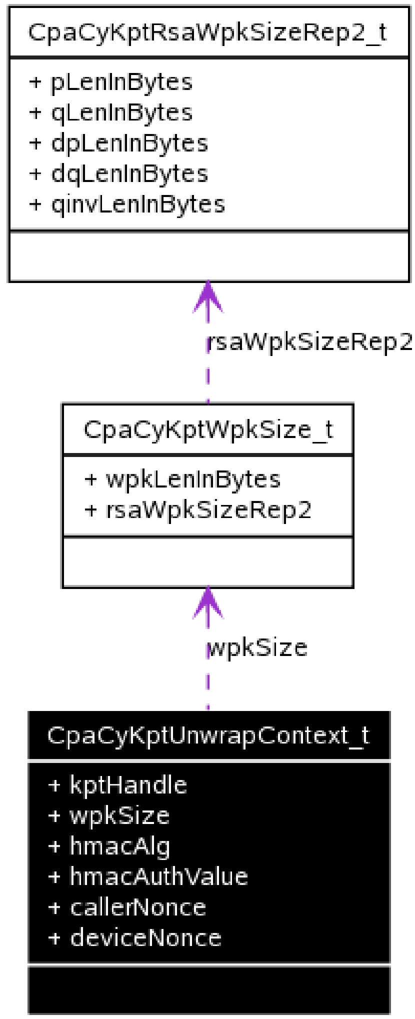
CpaCyKptRsaWpkSizeRep2 CpaCyKptWpkSize_t::rsaWpkSizeRep2

The byte length of wrapped private key for RSA rep2 case

22.7.4 CpaCyKptUnwrapContext_t Struct Reference

Collaboration diagram for CpaCyKptUnwrapContext_t:

22.7.4 CpaCyKptUnwrapContext_t Struct Reference



22.7.4.1 Detailed Description

File: cpa_cy_kpt.h

Structure of KPT unwrapping context.

This structure is a parameter of KPT crypto APIs, it contains data relating to KPT WPK unwrapping and HMAC authentication,application should complete those information in structure.

22.7.4.2 Data Fields

- **CpaCyKptHandle kptHandle**
- **CpaCyKptWpkSize wpkSize**
- **CpaCyKptHMACType hmacAlg**
- **Cpa8U hmacAuthValue** [CPA_CY_KPT_HMAC_LENGTH]
- **Cpa8U callerNonce** [CPA_CY_KPT_CALLER_NONCE_LENGTH]
- **Cpa8U deviceNonce** [CPA_CY_KPT_DEVICE_NONCE_LENGTH]

22.7.4.3 Field Documentation

CpaCyKptHandle CpaCyKptUnwrapContext_t::kptHandle

This is application's unique handle that identifies its (symmetric) wrapping key

CpaCyKptWpkSize CpaCyKptUnwrapContext_t::wpkSize

WPK's key size

CpaCyKptHMACType CpaCyKptUnwrapContext_t::hmacAlg

HMAC algorithm used in HMAC authentication in KPT crypto service

Cpa8U CpaCyKptUnwrapContext_t::hmacAuthValue[CPA_CY_KPT_HMAC_LENGTH]

HMAC authentication value input by the application in KPT crypto service;

Cpa8U CpaCyKptUnwrapContext_t::callerNonce[CPA_CY_KPT_CALLER_NONCE_LENGTH]

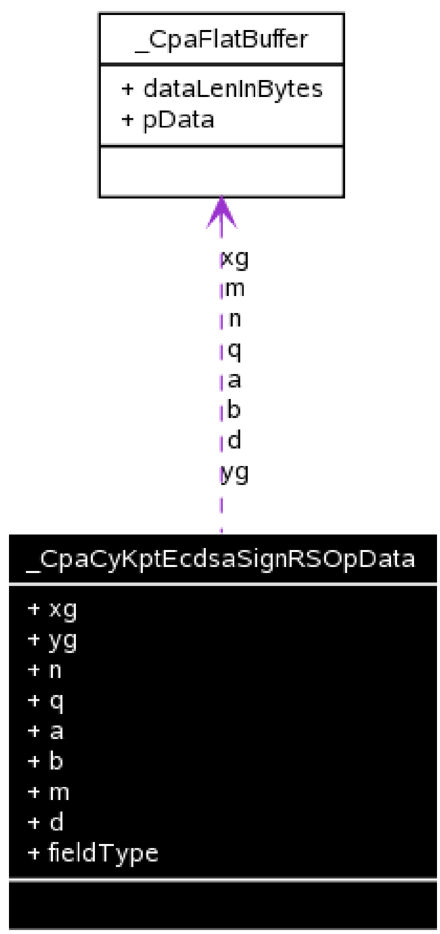
Caller(app) nonce generated by app in KPT crypto service

Cpa8U CpaCyKptUnwrapContext_t::deviceNonce[CPA_CY_KPT_DEVICE_NONCE_LENGTH]

Device nonce generated by device in KPT crypto service

22.7.5 _CpaCyKptEcdsaSignRSOpData Struct Reference

Collaboration diagram for _CpaCyKptEcdsaSignRSOpData:



22.7.5 _CpaCyKptEcdsaSignRSOpData Struct Reference

22.7.5.1 Detailed Description

File: cpa_cy_kpt.h

KPTECDSA Sign R & S Operation Data.

This structure contains the operation data for the cpaCyKptEcdsaSignRS function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyKptEcdsaSignRS function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyEcdsaSignRS()

22.7.5.2 Data Fields

- **CpaFlatBuffer xg**
- **CpaFlatBuffer yg**
- **CpaFlatBuffer n**
- **CpaFlatBuffer q**
- **CpaFlatBuffer a**
- **CpaFlatBuffer b**
- **CpaFlatBuffer m**
- **CpaFlatBuffer d**
- **CpaCyEcFieldType fieldType**

22.7.5.3 Field Documentation

CpaFlatBuffer _CpaCyKptEcdsaSignRSOpData::xg

x coordinate of base point G

CpaFlatBuffer _CpaCyKptEcdsaSignRSOpData::yg

y coordinate of base point G

CpaFlatBuffer _CpaCyKptEcdsaSignRSOpData::n

order of the base point G, which shall be prime

CpaFlatBuffer _CpaCyKptEcdsaSignRSOpData::q

prime modulus or irreducible polynomial over GF(2^r)

CpaFlatBuffer _CpaCyKptEcdsaSignRSOpData::a

a elliptic curve coefficient

CpaFlatBuffer _CpaCyKptEcdsaSignRSOpData::b

b elliptic curve coefficient

CpaFlatBuffer _CpaCyKptEcdsaSignRSOpData::m

digest of the message to be signed

CpaFlatBuffer _CpaCyKptEcdsaSignRSOpData::d

private key

CpaCyEcFieldType _CpaCyKptEcdsaSignRSOpData::fieldType

field type for the operation

22.8 Define Documentation

#define CPA_CY_KPT_MAX_IV_LENGTH

File: cpa_cy_kpt.h

Max length of initialization vector

Defines the permitted max iv length in bytes that may be used in private key wrapping/unwrapping.For AEC-GCM,iv length is 12 bytes,for AES-CBC,iv length is 16 bytes.

See also:
cpaCyKptWrappingFormat

#define CPA_CY_KPT_HMAC_LENGTH

File: cpa_cy_kpt.h

Max length of HMAC value in HMAC authentication during KPT crypto service.

Defines the permitted max HMAC value length in bytes that may be used to do HMAC verification in KPT crypto service.

See also:
cpaCyKptUnwrapContext

#define CPA_CY_KPT_CALLER_NONCE_LENGTH

File: cpa_cy_kpt.h

Length of nonce generated by application in HMAC authentication during KPT crypto service.

Defines the caller nonce length in bytes that will be used to do HMAC authentication in KPT crypto service.

See also:
cpaCyKptUnwrapContext

#define CPA_CY_KPT_DEVICE_NONCE_LENGTH

File: cpa_cy_kpt.h

Length of nonce generated by QAT in HMAC authentication during KPT crypto service.

Defines the device nonce length in bytes that will be used to do HMAC authentication in KPT crypto service.

See also:
cpaCyKptUnwrapContext

22.9 Typedef Documentation

typedef **Cpa64U CpaCyKptHandle**

File: cpa_cy_kpt.h

KPT wrapping key handle

Handle to a unique wrapping key in wrapping key table. Application creates it in KPT key transfer phase and maintains it for KPT Crypto service. For each KPT Crypto service API invocation,this handle will be used to get a SWK(Symmetric Wrapping Key) to unwrap WPK(Wrapped Private Key) before performing the requested crypto service.

typedef enum **CpaCyKptWrappingKeyType_t CpaCyKptWrappingKeyType**

File: cpa_cy_kpt.h

Cipher algorithms used to generate a wrapped private key (WPK) from the clear private key.

This enumeration lists supported cipher algorithms and modes.

typedef enum **CpaCyKptHMACType_t CpaCyKptHMACType**

File: cpa_cy_kpt.h

Hash algorithms used to generate WPK hash tag or used to do HMAC authentication in KPT crypto service.

This enumeration lists supported hash algorithms.

typedef enum **CpaCyKptKeyManagementStatus_t CpaCyKptKeyManagementStatus**

File: cpa_cy_kpt.h

Return Status

This enumeration lists all the possible return status after completing KPT APIs.

typedef enum **CpaCyKptKeySelectionFlags_t CpaCyKptKeySelectionFlags**

File: cpa_cy_kpt.h

Key selection flag.

This enumeration lists possible actions to be performed during cpaCyKptLoadKeys invocation.

typedef enum **CpaCyKptKeyAction_t CpaCyKptKeyAction**

File: cpa_cy_kpt.h

Key action.

PTT architecture support a "per-use" HMAC authorization for accessing and using key objects stored in PTT. This HMAC check is based on the use of running nonces shared between the application and PTT. To stay compatible with PTT's security protocol, QAT implements HMAC authorization protocol. This flag, set first time in cpaCyKptLoadKeys, will be used to determine whether HMAC authorization must be processed when QAT decrypts WPKs using SWKs.

typedef struct **CpaCyKptWrappingFormat_t CpaCyKptWrappingFormat**

File: cpa_cy_kpt.h

KPT wrapping format structure.

This structure defines wrapping format which is used to wrap clear private keys using a symmetric wrapping key.Application sets these parameters through the cpaCyKptLoadKeys calls.

typedef struct **CpaCyKptRsaWpkSizeRep2_t CpaCyKptRsaWpkSizeRep2**

File: cpa_cy_kpt.h

RSA wrapped private key size structure For Representation 2.

This structure contains byte length of wrapped quintuple of p,q,dP,dQ and qInv which are required for the second representation of RSA private key. PKCS #1 V2.1 specification defines the second representation of the RSA private key,The quintuple of p, q, dP, dQ, and qInv are required for this representation. **CpaCyRsaPrivateKeyRep2**

typedef union **CpaCyKptWpkSize_t CpaCyKptWpkSize**

File: cpa_cy_kpt.h

Wrapped private key size union.

A wrapped private key size union, either wrapped quintuple of RSA representation 2 private key,or byte length of wrapped ECC/RSA Rep1/DSA/ ECDSA private key.

typedef struct **CpaCyKptUnwrapContext_t CpaCyKptUnwrapContext**

File: cpa_cy_kpt.h

Structure of KPT unwrapping context.

This structure is a parameter of KPT crypto APIs, it contains data relating to KPT WPK unwrapping and HMAC authentication,application should complete those information in structure.

typedef struct **_CpaCyKptEcdsaSignRSOpData CpaCyKptEcdsaSignRSOpData**

File: cpa_cy_kpt.h

KPTECDSA Sign R & S Operation Data.

This structure contains the operation data for the cpaCyKptEcdsaSignRS function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

For optimal performance all data buffers SHOULD be 8-byte aligned.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyKptEcdsaSignRS function, and before it has been returned in the callback, undefined behavior will result.

See also:

cpaCyEcdsaSignRS()

22.10 Enumeration Type Documentation

enum **CpaCyKptWrappingKeyType_t**

File: cpa_cy_kpt.h

Cipher algorithms used to generate a wrapped private key (WPK) from the clear private key.

This enumeration lists supported cipher algorithms and modes.

enum **CpaCyKptHMACType_t**

File: cpa_cy_kpt.h

Hash algorithms used to generate WPK hash tag or used to do HMAC authentication in KPT crypto service.

This enumeration lists supported hash algorithms.

Enumerator:

CPA_CY_KPT_HMAC_TYPE_NULL No HMAC required

enum **CpaCyKptKeyManagementStatus_t**

File: cpa_cy_kpt.h

Return Status

This enumeration lists all the possible return status after completing KPT APIs.

Enumerator:

CPA_CY_KPT_SUCCESS
Generic success status for all KPT wrapping key handling functions
CPA_CY_KPT_REGISTER_HANDLE_FAIL_RETRY
WKT is busy, retry after some time

CPA_CY_KPT_REGISTER_HANDLE_FAIL_DUPLICATE
Handle is already present in WKT; this is attempt at duplication
CPA_CY_KPT_LOAD_KEYS_FAIL_INVALID_HANDLE
LoadKey call does not provide a handle that was previously registered. Either application error, or malicious application. Reject request to load the key.
CPA_CY_KPT_REGISTER_HANDLE_FAIL_WKT_FULL
Failed to register wrapping key as WKT is full
CPA_CY_KPT_WKT_ENTRY_NOT_FOUND
Unable to find SWK entry by handle
CPA_CY_KPT_REGISTER_HANDLE_FAIL_INSTANCE_QUOTA_EXCEEDED
This application has opened too many WKT entries. A Quota is enforced to prevent DoS attacks
CPA_CY_KPT_LOADKEYS_FAIL_CHECKSUM_ERROR
Checksum error in key loading
CPA_CY_KPT_LOADKEYS_FAIL_HANDLE_NOT_REGISTERED
Key is not registered in key loading
CPA_CY_KPT_LOADKEYS_FAIL_POSSIBLE_DOS_ATTACK
Possible Dos attack happened in key loading
CPA_CY_KPT_LOADKEYS_FAIL_INVALID_AC_SEND_HANDLE
Invalid key handle got from PTT
CPA_CY_KPT_LOADKEYS_FAIL_INVALID_DATA_OBJ
Invalid data object got from PTT

enum **CpaCyKptKeySelectionFlags_t**

File: cpa_cy_kpt.h

Key selection flag.

This enumeration lists possible actions to be performed during cpaCyKptLoadKeys invocation.

Enumerator:

CPA_CY_KPT_SWK	Symmetric wrapping key,only a SWK will be loaded from PTT to QAT
CPA_CY_KPT_WPK	Wrapped private key, a data blob including SWK and CPK will be loaded from PTT to QAT, and WPK will be return to application.
CPA_CY_KPT_OPAQUE_DATA	Opaque data,a opaque data will be loaded from PTT to QAT
CPA_CY_KPT_HMAC_AUTH_PARAMS	HMAC auth params,HMAC auth params will be loaded from PTT to QAT
CPA_CY_KPT_RN_SEED	DRBG seed,A rondom data generated by PTT will be loaded from PTT to QAT

enum **CpaCyKptKeyAction_t**

File: cpa_cy_kpt.h

Key action.

PTT architecture support a "per-use" HMAC authorization for accessing and using key objects stored in PTT. This HMAC check is based on the use of running nonces shared between the application and PTT. To stay compatible with PTT's security protocol, QAT implements HMAC authorization protocol. This flag, set first time in cpaCyKptLoadKeys, will be used to determine whether HMAC authorization must be processed when QAT decrypts WPKs using SWKs.

Enumerator:

CPA_CY_KPT_NO_HMAC_AUTH_CHECK	Do not need HMAC authentication check in KPT Crypto service
CPA_CY_KPT_HMAC_AUTH_CHECK	Need HMAC authentication check in KPT Crypto service

22.11 Function Documentation

CpaStatus cpaCyKptRegisterKeyHandle (**CpaInstanceHandle** *instanceHandle*, **CpaCyKptHandle** *keyHandle*, **CpaCyKptKeyManagementStatus** * *pKptStatus*)

File: cpa_cy_kpt.h

Perform KPT key handle register function.

Used for loading an application's wrapping key from PTT to QAT. An application first precomputes/initializes a 64 bit handle value using CPU based RDRAND instruction or other means and passes it to QAT. This will signal to QAT that a KPT key transfer operation is about to begin

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	QAT service instance handle.
`[in]`	*keyHandle*	A 64-bit handle value
`[out]`	*pKptStatus*	One of the status codes denoted in the enumerate type of cpaCyKptKeyManagementStatus

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.

CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
This function operates in a synchronous manner and no asynchronous callback will be generated.

See also:
None

```
CpaStatus cpaCyKptLoadKeys ( CpaInstanceHandle               instanceHandle,
                             CpaCyKptHandle                  keyHandle,
                             CpaCyKptWrappingFormat *        pKptWrappingFormat,
                             CpaCyKptKeySelectionFlags       keySelFlag,
                             CpaCyKptKeyAction               keyAction,
                             CpaFlatBuffer *                 pOutputData,
                             CpaCyKptKeyManagementStatus *   pKptStatus
                           )
```

File: cpa_cy_kpt.h

Perform KPT key loading function.

This function is invoked by QAT application after instructing PTT to send its wrapping key to QAT. After PTT returns a TPM_SUCCESS, the wrapping key structure is placed in QAT. The Application completes the 3-way handshake by invoking this API and requesting QAT to store the wrapping key, along with its handle.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	QAT service instance handle.
[in]	*keyHandle*	A 64-bit handle value

[in]	*keySelFlag*	Flag to indicate which kind of mode (SWK or WPK) should be loaded.
[in]	*keyAction*	Whether HAMC authentication is needed
[in]	*pKptWrappingFormat*	Pointer to CpaCyKptWrappingFormat whose fields will be written to WKT.
[out]	*pOutputData*	FlatBuffer pointer, which contains the wrapped private key structure used by application.
[out]	*pKptStatus*	One of the status codes denoted in the enumerate type CpaCyKptKeyManagementStatus

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
None

CpaStatus cpaCyKptDeleteKey (	**CpaInstanceHandle**	*instanceHandle*,
	CpaCyKptHandle	*keyHandle*,
	CpaCyKptKeyManagementStatus *	*pkptstatus*
)		

File: cpa_cy_kpt.h

Perform KPT delete keys function according to key handle

Before closing a QAT session(instance), an application that has previously stored its wrapping key in QAT using the KPT framework executes this call to delete its wrapping key in QAT.

Context:
This is a synchronous function and it can sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
This function is synchronous and blocking.

Reentrant:

No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	QAT service instance handle.
`[in]`	*keyHandle*	A 64-bit handle value
`[out]`	*pkptstatus*	One of the status codes denoted in the enumerate type CpaCyKptKeyManagementStatus

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.

Precondition:
Component has been initialized.

Postcondition:
None

Note:
None

See also:
None

```
CpaStatus cpaCyKptRsaDecrypt ( const CpaInstanceHandle        instanceHandle,
                               const CpaCyGenFlatBufCbFunc    pRsaDecryptCb,
                               void *                         pCallbackTag,
                               const CpaCyRsaDecryptOpData *  pDecryptOpData,
                               CpaFlatBuffer *                pOutputData,
                               CpaFlatBuffer *                pKptUnwrapContext
                             )
```

File: cpa_cy_kpt.h

Perform KPT mode RSA decrypt primitive operation on the input data.

This function is variant of cpaCyRsaDecrypt, which will perform an RSA decryption primitive operation on the input data using the specified RSA private key which are encrypted. As the RSA decryption primitive and signing primitive operations are mathematically identical this function may also be used to perform an RSA signing primitive operation.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pRsaDecryptCb*	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	Opaque User Data for this specific call. Will be returned unchanged in the callback.
`[in]`	*pDecryptOpData*	Structure containing all the data needed to perform the RSA decrypt operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pOutputData*	Pointer to structure into which the result of the RSA decryption primitive is written. The client MUST allocate this memory. The data pointed to is an integer in big-endian order. The value will be between 0 and the modulus n - 1. On invocation the callback function will contain this parameter in the pOut parameter.
`[in]`	*pKptUnwrapContext*	Pointer of structure into which the content of KptUnwrapContext is kept,The client MUST allocate this memory and copy structure KptUnwrapContext into this flat buffer.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting.Resubmit the request.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
By virtue of invoking cpaSyKptRsaDecrypt, the implementation understands that pDecryptOpData contains an encrypted private key that requires unwrapping. KptUnwrapContext contains an 'KptHandle' field that points to the unwrapping key in the WKT. When pRsaDecryptCb is non-NULL an asynchronous callback is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. For optimal performance, data pointers SHOULD be 8-byte aligned. In KPT release,private key field in CpaCyRsaDecryptOpData is a concatenation of cipher text and hash tag. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyRsaDecryptOpData, **CpaCyGenFlatBufCbFunc**, **cpaCyRsaGenKey()**, **cpaCyRsaEncrypt()**

CpaStatus cpaCyKptEcdsaSignRS (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyEcdsaSignRSCbFunc**	*pCb*,
	void *	*pCallbackTag*,
	const **CpaCyKptEcdsaSignRSOpData** *	*pOpData*,
	CpaBoolean *	*pSignStatus*,
	CpaFlatBuffer *	*pR*,
	CpaFlatBuffer *	*pS*,
	CpaFlatBuffer *	*pKptUnwrapContext*
)		

File: cpa_cy_kpt.h

Generate ECDSA Signature R & S.

This function is a varient of cpaCyEcdsaSignRS, it generates ECDSA Signature R & S as per ANSI X9.62 2005 section 7.3.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pSignStatus*	In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
[out]	*pR*	ECDSA message signature r.
[out]	*pS*	ECDSA message signature s.
[in]	*pKptUnwrapContext*	

Pointer of structure into which the content of KptUnwrapContext is kept,The client MUST allocate this memory and copy structure KptUnwrapContext into this flat buffer.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:

The component has been initialized via cpaCyStartInstance function.

Postcondition:

None

Note:

By virtue of invoking the cpaCyKptEcdsaSignRS, the implementation understands CpaCyEcdsaSignRSOpData contains an encrypted private key that requires unwrapping. KptUnwrapContext contains an 'KptHandle' field that points to the unwrapping key in the WKT. When pCb is non-NULL an asynchronous callback of type CpaCyEcdsaSignRSCbFunc generated in response to this function call. In KPT release,private key field in CpaCyEcdsaSignRSOpData is a concatenation of cipher text and hash tag.

See also:

None

CpaStatus cpaCyKptDsaSignS (	const **CpaInstanceHandle**	*instanceHandle*,
	const **CpaCyDsaGenCbFunc**	*pCb*,
	void *	*pCallbackTag*,
	const **CpaCyDsaSSignOpData** *	*pOpData*,
	CpaBoolean *	*pProtocolStatus*,
	CpaFlatBuffer *	*pS*,
	CpaFlatBuffer *	*pKptUnwrapContext*
)		

File: cpa_cy_kpt.h

This function is varient of cpaCyDsaSignS,which generate DSA S Signature.

This function generates the DSA S signature as described in FIPS 186-3 Section 4.6: s = (k^-1(z + xr)) mod q

Here, z = the leftmost min(N, outlen) bits of Hash(M). This function does not perform the SHA digest; z is computed by the caller and passed as a parameter in the pOpData field.

The protocol status, returned in the callback function as parameter protocolStatus (or, in the case of synchronous invocation, in the parameter *pProtocolStatus) is used to indicate whether the value s == 0.

Specifically, (protocolStatus == CPA_TRUE) means s != 0, while (protocolStatus == CPA_FALSE) means s == 0.

If signature r has been generated in advance, then this function can be used to generate the signature s once the message becomes available.

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

`[in]`	*instanceHandle*	Instance handle.
`[in]`	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
`[in]`	*pCallbackTag*	User-supplied value to help identify request.
`[in]`	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
`[out]`	*pProtocolStatus*	The result passes/fails the DSA protocol related checks.
`[out]`	*pS*	DSA message signature s. On invocation the callback function will contain this parameter in the pOut parameter.
`[in]`	*pKptUnwrapContext*	Pointer of structure into which the content of KptUnwrapContext is kept,The client MUST allocate this memory and copy structure KptUnwrapContext into this flat buffer.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:

When pCb is non-NULL an asynchronous callback of type CpaCyDsaSSignCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

By virtue of invoking cpaCyKptDsaSignS, the implementation understands CpaCyDsaSSignOpData contains an encrypted private key that requires unwrapping. KptUnwrapContext contains an 'KptHandle' field that points to the unwrapping key in the WKT. In KPT,private key field in CpaCyDsaSSignOpData is a concatenation of cipher text and hash tag. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyDsaSSignOpData, **CpaCyDsaGenCbFunc**, **cpaCyDsaSignR()**, **cpaCyDsaSignRS()**

CpaStatus cpaCyKptDsaSignRS	(	const **CpaInstanceHandle**	*instanceHandle*,
		const **CpaCyDsaRSSignCbFunc**	*pCb*,
		void *	*pCallbackTag*,
		const **CpaCyDsaRSSignOpData** *	*pOpData*,
		CpaBoolean *	*pProtocolStatus*,
		CpaFlatBuffer *	*pR*,
		CpaFlatBuffer *	*pS*,
		CpaFlatBuffer *	*pKptUnwrapContext*
	)		

File: cpa_cy_kpt.h

This function is a varient of cpaCyDsaSignRS,which generate DSA R and S Signature

This function generates the DSA R and S signatures as described in FIPS 186-3 Section 4.6:

r = (g^k mod p) mod q s = (k^-1(z + xr)) mod q

Here, z = the leftmost min(N, outlen) bits of Hash(M). This function does not perform the SHA digest; z is computed by the caller and passed as a parameter in the pOpData field.

The protocol status, returned in the callback function as parameter protocolStatus (or, in the case of synchronous invocation, in the parameter *pProtocolStatus) is used to indicate whether either of the values r or s are zero.

Specifically, (protocolStatus == CPA_TRUE) means neither is zero (i.e. (r != 0) && (s != 0)), while (protocolStatus == CPA_FALSE) means that at least one of r or s is zero (i.e. (r == 0) || (s == 0)).

Context:
When called as an asynchronous function it cannot sleep. It can be executed in a context that does not permit sleeping. When called as a synchronous function it may sleep. It MUST NOT be executed in a context that DOES NOT permit sleeping.

Assumptions:
None

Side-Effects:
None

Blocking:
Yes when configured to operate in synchronous mode.

Reentrant:
No

Thread-safe:
Yes

Parameters:

[in]	*instanceHandle*	Instance handle.
[in]	*pCb*	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	*pCallbackTag*	User-supplied value to help identify request.
[in]	*pOpData*	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	*pProtocolStatus*	The result passes/fails the DSA protocol related checks.
[out]	*pR*	DSA message signature r.
[out]	*pS*	DSA message signature s.
[in]	*pKptUnwrapContext*	Pointer of structure into which the content of KptUnwrapContext is kept,The client MUST allocate this memory and copy structure KptUnwrapContext into this flat buffer.

Return values:

CPA_STATUS_SUCCESS	Function executed successfully.
CPA_STATUS_FAIL	Function failed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_RESTARTING	API implementation is restarting. Resubmit the request.
CPA_STATUS_UNSUPPORTED	Function is not supported.

Precondition:
The component has been initialized via cpaCyStartInstance function.

Postcondition:
None

Note:
When pCb is non-NULL an asynchronous callback of type CpaCyDsaRSSignCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned. By virtue of invoking CyKptDsaSignRS, the implementation understands CpaCyDsaRSSignOpData contains an enrypted private key that requires unwrapping. KptUnwrapContext contains an 'KptHandle' field that points to the unwrapping key in the WKT. In KPT,private key field in CpaCyDsaRSSignOpData is a concatenation of cipher text and hash tag. For optimal performance, data pointers SHOULD be 8-byte aligned.

See also:
CpaCyDsaRSSignOpData, **CpaCyDsaRSSignCbFunc**, **cpaCyDsaSignR()**, **cpaCyDsaSignS()**