API Reference Manual

Intel® QuickAssist Technology

Hardware Version 2.0

June 2023

Document Number: 330685

Contents

1 API Description

1.1 Classes and Structs

1.1.1 Struct _CpaBufferList

- Defined in file_api_cpa.h

Struct Documentation

struct `_CpaBufferList`

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

Description: 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.

Public Members

Cpa32U `numBuffers`

Number of buffers in the list

CpaFlatBuffer *`pBuffers`

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

void *`pUserData`

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

void *`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.

1.1.2 Struct _CpaCrcControlData

- Defined in file_api_cpa.h

Struct Documentation

struct `_CpaCrcControlData`

Crc Control data structure for programmable CRC engine.

Description: This structure specifies CRC algorithm parameters which are used to configure a programmable CRC engine. It can be used to specify a CRC algorithm, other than those natively supported by the API.

Public Members

Cpa64U `polynomial`

Polynomial used for CRC64 calculation.

Cpa64U `initialValue`

Initial value to be used for seeding the CRC.

CpaBoolean `reflectIn`

Reflect bit order before CRC calculation.

CpaBoolean `reflectOut`

Reflect bit order after CRC calculation.

Cpa64U `xorOut`

XOR pattern to XOR with the final CRC residue after any output reflection

1.1.3 Struct _CpaCrcData

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaCrcData`

Collection of CRC related data

Description: This structure contains data facilitating CRC calculations. After successful request, this structure will contain all resulting CRCs. Integrity specific CRCs (when enabled/supported) are located in 'CpaIntegrityCrc integrityCrc' field for 32bit values and in 'CpaIntegrityCrc64b integrityCrC64b' field for 64 bit values. Integrity CRCs cannot be accumulated across multiple requests and do not provide seeding capabilities.

Note: this structure must be allocated in physical contiguous memory

Public Members

Cpa32U `crc32`

CRC32 calculated on the input buffer during compression requests and on the output buffer during decompression requests.

Cpa32U `adler32`

Adler32 calculated on the input buffer during compression requests and on the output buffer during decompression requests.

CpaIntegrityCrc `integrityCrc`

32bit Integrity CRCs

CpaIntegrityCrc64b `integrityCrc64b`

64bit Integrity CRCs

1.1.4 Struct _CpaCyCapabilitiesInfo

- Defined in file_api_lac_cpa_cy_im.h

Struct Documentation

struct `_CpaCyCapabilitiesInfo`

Cryptographic Capabilities Info

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.

Description: 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.

Public Members

CpaBoolean `symSupported`

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

CpaBoolean `symDpSupported`

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

CpaBoolean `dhSupported`

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

CpaBoolean `dsaSupported`

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

CpaBoolean `rsaSupported`

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

CpaBoolean `ecSupported`

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

CpaBoolean `ecdhSupported`

CPA_TRUE if instance supports the Elliptic Curve Diffie Hellman API. See cpaCyEcdh.

CpaBoolean `ecdsaSupported`

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

CpaBoolean `keySupported`

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

CpaBoolean `lnSupported`

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

CpaBoolean `primeSupported`

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

CpaBoolean `drbgSupported`

CPA_TRUE if instance supports the DRBG API. See cpaCyDrbg.

CpaBoolean `nrbgSupported`

CPA_TRUE if instance supports the NRBG API. See cpaCyNrbg.

CpaBoolean `randSupported`

CPA_TRUE if instance supports the random bit/number generation API. See cpaCyRand.

CpaBoolean `kptSupported`

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

CpaBoolean `hkdfSupported`

CPA_TRUE if instance supports the HKDF components of the KeyGen API. See Cryptographic Key and Mask Generation API.

CpaBoolean `extAlgchainSupported`

CPA_TRUE if instance supports algorithm chaining for certain wireless algorithms. Please refer to implementation for details. See Symmetric Cipher and Hash Cryptographic API.

CpaBoolean `ecEdMontSupported`

CPA_TRUE if instance supports the Edwards and Montgomery elliptic curves of the EC API. See Elliptic Curve (EC) API

CpaBoolean `ecSm2Supported`

CPA_TRUE if instance supports the EcSM2 API. See cpaCyEcsm2.

1.1.5 Struct _CpaCyDhPhase1KeyGenOpData

- Defined in file_api_lac_cpa_cy_dh.h

Struct Documentation

struct `_CpaCyDhPhase1KeyGenOpData`

Diffie-Hellman Phase 1 Key Generation Data.

Description: 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.

Public Members

CpaFlatBuffer `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, 4096 or 8192.

CpaFlatBuffer `baseG`

Flat buffer containing a pointer to base (g). This MUST comply with the following: 0 < g < p.

CpaFlatBuffer `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.

1.1.6 Struct _CpaCyDhPhase2SecretKeyGenOpData

- Defined in file_api_lac_cpa_cy_dh.h

Struct Documentation

struct `_CpaCyDhPhase2SecretKeyGenOpData`

Diffie-Hellman Phase 2 Secret Key Generation Data.

Description: 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.

Public Members

CpaFlatBuffer `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, 4096 or 8192. This SHOULD be same prime number as was used in the phase 1 key generation operation.

CpaFlatBuffer `remoteOctetStringPV`

Flat buffer containing a pointer to the remote entity octet string Public Value (PV).

CpaFlatBuffer `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).

1.1.7 Struct _CpaCyDhStats

- Defined in file_api_lac_cpa_cy_dh.h

Struct Documentation

struct `_CpaCyDhStats`

Diffie-Hellman Statistics.

Deprecated:

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

Description: This structure contains statistics on the Diffie-Hellman operations. Statistics are set to zero when the component is initialized, and are collected per instance.

Public Members

Cpa32U `numDhPhase1KeyGenRequests`

Total number of successful Diffie-Hellman phase 1 key generation requests.

Cpa32U `numDhPhase1KeyGenRequestErrors`

Total number of Diffie-Hellman phase 1 key generation requests that had an error and could not be processed.

Cpa32U `numDhPhase1KeyGenCompleted`

Total number of Diffie-Hellman phase 1 key generation operations that completed successfully.

Cpa32U `numDhPhase1KeyGenCompletedErrors`

Total number of Diffie-Hellman phase 1 key generation operations that could not be completed successfully due to errors.

Cpa32U `numDhPhase2KeyGenRequests`

Total number of successful Diffie-Hellman phase 2 key generation requests.

Cpa32U `numDhPhase2KeyGenRequestErrors`

Total number of Diffie-Hellman phase 2 key generation requests that had an error and could not be processed.

Cpa32U `numDhPhase2KeyGenCompleted`

Total number of Diffie-Hellman phase 2 key generation operations that completed successfully.

Cpa32U `numDhPhase2KeyGenCompletedErrors`

Total number of Diffie-Hellman phase 2 key generation operations that could not be completed successfully due to errors.

1.1.8 Struct _CpaCyDhStats64

- Defined in file_api_lac_cpa_cy_dh.h

Struct Documentation

struct `_CpaCyDhStats64`

Diffie-Hellman Statistics (64-bit version).

Description: 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.

Public Members

Cpa64U `numDhPhase1KeyGenRequests`

Total number of successful Diffie-Hellman phase 1 key generation requests.

Cpa64U `numDhPhase1KeyGenRequestErrors`

Total number of Diffie-Hellman phase 1 key generation requests that had an error and could not be processed.

Cpa64U `numDhPhase1KeyGenCompleted`

Total number of Diffie-Hellman phase 1 key generation operations that completed successfully.

Cpa64U `numDhPhase1KeyGenCompletedErrors`

Total number of Diffie-Hellman phase 1 key generation operations that could not be completed successfully due to errors.

Cpa64U `numDhPhase2KeyGenRequests`

Total number of successful Diffie-Hellman phase 2 key generation requests.

Cpa64U `numDhPhase2KeyGenRequestErrors`

Total number of Diffie-Hellman phase 2 key generation requests that had an error and could not be processed.

Cpa64U `numDhPhase2KeyGenCompleted`

Total number of Diffie-Hellman phase 2 key generation operations that completed successfully.

Cpa64U `numDhPhase2KeyGenCompletedErrors`

Total number of Diffie-Hellman phase 2 key generation operations that could not be completed successfully due to errors.

1.1.9 Struct _CpaCyDsaGParamGenOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Struct Documentation

struct `_CpaCyDsaGParamGenOpData`

DSA G Parameter Generation Operation Data.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

Description: 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 numbers MUST be stored in big-endian order.

See also:

cpaCyDsaGenGParam()

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.

Public Members

CpaFlatBuffer `P`

DSA group parameter p

CpaFlatBuffer `Q`

DSA group parameter q

CpaFlatBuffer `H`

any integer with 1 < h < p - 1

1.1.10 Struct _CpaCyDsaPParamGenOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Struct Documentation

struct `_CpaCyDsaPParamGenOpData`

DSA P Parameter Generation Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. X.pData[0] = MSB.

See also:

cpaCyDsaGenPParam()

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.

Public Members

CpaFlatBuffer `X`

2^(L-1) <= X < 2^L (from FIPS 186-3)

CpaFlatBuffer `Q`

DSA group parameter q

1.1.11 Struct _CpaCyDsaRSignOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Struct Documentation

struct `_CpaCyDsaRSignOpData`

DSA R Sign Operation Data.

For optimal performance all data SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

See also:

cpaCyDsaSignR()

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.

Public Members

CpaFlatBuffer `P`

DSA group parameter p

CpaFlatBuffer `Q`

DSA group parameter q

CpaFlatBuffer `G`

DSA group parameter g

CpaFlatBuffer `K`

DSA secret parameter k for signing

1.1.12 Struct _CpaCyDsaRSSignOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Struct Documentation

struct `_CpaCyDsaRSSignOpData`

DSA R & S Sign Operation Data.

For optimal performance all data SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

See also:

cpaCyDsaSignRS()

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.

Public Members

CpaFlatBuffer `P`
DSA group parameter p

CpaFlatBuffer `Q`
DSA group parameter q

CpaFlatBuffer `G`
DSA group parameter g

CpaFlatBuffer `X`
DSA private key x

CpaFlatBuffer `K`
DSA secret parameter k for signing

CpaFlatBuffer `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

1.1.13 Struct _CpaCyDsaSSignOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Struct Documentation

struct `_CpaCyDsaSSignOpData`

DSA S Sign Operation Data.

For optimal performance all data SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. Q.pData[0] = MSB.

See also:

cpaCyDsaSignS()

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.

Public Members

CpaFlatBuffer Q

DSA group parameter q

CpaFlatBuffer X

DSA private key x

CpaFlatBuffer K

DSA secret parameter k for signing

CpaFlatBuffer R

DSA message signature r

CpaFlatBuffer 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

1.1.14 Struct _CpaCyDsaStats

- Defined in file_api_lac_cpa_cy_dsa.h

Struct Documentation

struct `_CpaCyDsaStats`

Cryptographic DSA Statistics.

Deprecated:

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

Description: This structure contains statistics on the Cryptographic DSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

Public Members

Cpa32U `numDsaPParamGenRequests`

Total number of successful DSA P parameter generation requests.

Cpa32U `numDsaPParamGenRequestErrors`

Total number of DSA P parameter generation requests that had an error and could not be processed.

Cpa32U `numDsaPParamGenCompleted`

Total number of DSA P parameter generation operations that completed successfully.

Cpa32U `numDsaPParamGenCompletedErrors`

Total number of DSA P parameter generation operations that could not be completed successfully due to errors.

Cpa32U `numDsaGParamGenRequests`

Total number of successful DSA G parameter generation requests.

Cpa32U `numDsaGParamGenRequestErrors`

Total number of DSA G parameter generation requests that had an error and could not be processed.

Cpa32U `numDsaGParamGenCompleted`

Total number of DSA G parameter generation operations that completed successfully.

Cpa32U `numDsaGParamGenCompletedErrors`

Total number of DSA G parameter generation operations that could not be completed successfully due to errors.

Cpa32U `numDsaYParamGenRequests`

Total number of successful DSA Y parameter generation requests.

Cpa32U `numDsaYParamGenRequestErrors`

Total number of DSA Y parameter generation requests that had an error and could not be processed.

Cpa32U `numDsaYParamGenCompleted`

Total number of DSA Y parameter generation operations that completed successfully.

Cpa32U `numDsaYParamGenCompletedErrors`

Total number of DSA Y parameter generation operations that could not be completed successfully due to errors.

Cpa32U `numDsaRSignRequests`

Total number of successful DSA R sign generation requests.

Cpa32U `numDsaRSignRequestErrors`

Total number of DSA R sign requests that had an error and could not be processed.

Cpa32U `numDsaRSignCompleted`

Total number of DSA R sign operations that completed successfully.

Cpa32U `numDsaRSignCompletedErrors`

Total number of DSA R sign operations that could not be completed successfully due to errors.

Cpa32U `numDsaSSignRequests`

Total number of successful DSA S sign generation requests.

Cpa32U `numDsaSSignRequestErrors`

Total number of DSA S sign requests that had an error and could not be processed.

Cpa32U `numDsaSSignCompleted`

Total number of DSA S sign operations that completed successfully.

Cpa32U `numDsaSSignCompletedErrors`

Total number of DSA S sign operations that could not be completed successfully due to errors.

Cpa32U `numDsaRSSignRequests`

Total number of successful DSA RS sign generation requests.

Cpa32U `numDsaRSSignRequestErrors`

Total number of DSA RS sign requests that had an error and could not be processed.

Cpa32U `numDsaRSSignCompleted`

Total number of DSA RS sign operations that completed successfully.

Cpa32U `numDsaRSSignCompletedErrors`

Total number of DSA RS sign operations that could not be completed successfully due to errors.

Cpa32U `numDsaVerifyRequests`

Total number of successful DSA verify generation requests.

Cpa32U `numDsaVerifyRequestErrors`

Total number of DSA verify requests that had an error and could not be processed.

Cpa32U `numDsaVerifyCompleted`

Total number of DSA verify operations that completed successfully.

Cpa32U `numDsaVerifyCompletedErrors`

Total number of DSA verify operations that could not be completed successfully due to errors.

Cpa32U `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.

1.1.15 Struct _CpaCyDsaStats64

- Defined in file_api_lac_cpa_cy_dsa.h

Struct Documentation

struct `_CpaCyDsaStats64`

Cryptographic DSA Statistics (64-bit version).

Description: 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.

Public Members

Cpa64U `numDsaPParamGenRequests`

Total number of successful DSA P parameter generation requests.

Cpa64U `numDsaPParamGenRequestErrors`

Total number of DSA P parameter generation requests that had an error and could not be processed.

Cpa64U `numDsaPParamGenCompleted`

Total number of DSA P parameter generation operations that completed successfully.

Cpa64U `numDsaPParamGenCompletedErrors`

Total number of DSA P parameter generation operations that could not be completed successfully due to errors.

Cpa64U `numDsaGParamGenRequests`

Total number of successful DSA G parameter generation requests.

Cpa64U `numDsaGParamGenRequestErrors`

Total number of DSA G parameter generation requests that had an error and could not be processed.

Cpa64U `numDsaGParamGenCompleted`

Total number of DSA G parameter generation operations that completed successfully.

Cpa64U `numDsaGParamGenCompletedErrors`

Total number of DSA G parameter generation operations that could not be completed successfully due to errors.

Cpa64U `numDsaYParamGenRequests`

Total number of successful DSA Y parameter generation requests.

Cpa64U `numDsaYParamGenRequestErrors`

Total number of DSA Y parameter generation requests that had an error and could not be processed.

Cpa64U `numDsaYParamGenCompleted`

Total number of DSA Y parameter generation operations that completed successfully.

Cpa64U `numDsaYParamGenCompletedErrors`

Total number of DSA Y parameter generation operations that could not be completed successfully due to errors.

Cpa64U `numDsaRSignRequests`

Total number of successful DSA R sign generation requests.

Cpa64U `numDsaRSignRequestErrors`

Total number of DSA R sign requests that had an error and could not be processed.

Cpa64U `numDsaRSignCompleted`

Total number of DSA R sign operations that completed successfully.

Cpa64U `numDsaRSignCompletedErrors`

Total number of DSA R sign operations that could not be completed successfully due to errors.

Cpa64U `numDsaSSignRequests`

Total number of successful DSA S sign generation requests.

Cpa64U `numDsaSSignRequestErrors`

Total number of DSA S sign requests that had an error and could not be processed.

Cpa64U `numDsaSSignCompleted`

Total number of DSA S sign operations that completed successfully.

Cpa64U `numDsaSSignCompletedErrors`

Total number of DSA S sign operations that could not be completed successfully due to errors.

Cpa64U `numDsaRSSignRequests`

Total number of successful DSA RS sign generation requests.

Cpa64U `numDsaRSSignRequestErrors`

Total number of DSA RS sign requests that had an error and could not be processed.

Cpa64U `numDsaRSSignCompleted`

Total number of DSA RS sign operations that completed successfully.

Cpa64U `numDsaRSSignCompletedErrors`

Total number of DSA RS sign operations that could not be completed successfully due to errors.

Cpa64U `numDsaVerifyRequests`

Total number of successful DSA verify generation requests.

Cpa64U `numDsaVerifyRequestErrors`

Total number of DSA verify requests that had an error and could not be processed.

Cpa64U `numDsaVerifyCompleted`

Total number of DSA verify operations that completed successfully.

Cpa64U `numDsaVerifyCompletedErrors`

Total number of DSA verify operations that could not be completed successfully due to errors.

Cpa64U `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.

1.1.16 Struct _CpaCyDsaVerifyOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Struct Documentation

struct `_CpaCyDsaVerifyOpData`

DSA Verify Operation Data.

For optimal performance all data SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

See also:

cpaCyDsaVerify()

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.

Public Members

CpaFlatBuffer **P**

DSA group parameter p

CpaFlatBuffer **Q**

DSA group parameter q

CpaFlatBuffer **G**

DSA group parameter g

CpaFlatBuffer **Y**

DSA public key y

CpaFlatBuffer **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 **R**

DSA message signature r

CpaFlatBuffer **S**

DSA message signature s

1.1.17 Struct _CpaCyDsaYParamGenOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Struct Documentation

struct `_CpaCyDsaYParamGenOpData`

DSA Y Parameter Generation Operation Data.

For optimal performance all data SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

See also:

cpaCyDsaGenYParam()

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.

Public Members

CpaFlatBuffer `P`

DSA group parameter p

CpaFlatBuffer `G`

DSA group parameter g

CpaFlatBuffer `X`

DSA private key x

1.1.18 Struct _CpaCyEcCurve

- Defined in file_api_lac_cpa_cy_ec.h

Struct Documentation

struct `_CpaCyEcCurve`

Unified curve parameters.

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.

Description: This structure provides a single data type that can describe a number of different curve types. The intention is to add further curve types in the future, thus the union field will allow for that expansion.

For optimal performance all data buffers SHOULD be 8-byte aligned.

See also:

CpaCyEcCurveParameters cpaCyEcGenericPointMultiply cpaCyEcGenericPointVerify

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

Public Members

CpaCyEcCurveType `curveType`

CpaCyEcCurveParameters `parameters`

1.1.19 Struct _CpaCyEcCurveParametersWeierstrass

- Defined in file_api_lac_cpa_cy_ec.h

Struct Documentation

struct `_CpaCyEcCurveParametersWeierstrass`

Curve parameters for a Weierstrass type curve.

For optimal performance all data buffers SHOULD be 8-byte aligned. The legend used in this structure is borrowed from RFC7748

Description: This structure contains curve parameters for Weierstrass type curve: y^2 = x^3 + ax + b 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 function.

See also:

CpaCyEcCurveParameters CpaCyEcFieldType

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

Public Members

CpaCyEcFieldType `fieldType`

Prime or Binary

CpaFlatBuffer `p`

Prime modulus or irreducible polynomial over GF(2^m)

CpaFlatBuffer `a`

a coefficient

CpaFlatBuffer `b`

b coefficient

CpaFlatBuffer `h`

Cofactor

1.1.20 Struct _CpaCyEcdsaSignROpData

- Defined in file_api_lac_cpa_cy_ecdsa.h

Struct Documentation

struct `_CpaCyEcdsaSignROpData`

ECDSA Sign R Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

cpaCyEcdsaSignR()

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.

Public Members

CpaFlatBuffer `xg`

x coordinate of base point G

CpaFlatBuffer `yg`

y coordinate of base point G

CpaFlatBuffer `n`

order of the base point G, which shall be prime

CpaFlatBuffer `q`

prime modulus or irreducible polynomial over GF(2^r)

CpaFlatBuffer `a`

a elliptic curve coefficient

CpaFlatBuffer `b`

b elliptic curve coefficient

CpaFlatBuffer `k`

random value (k > 0 and k < n)

CpaCyEcFieldType `fieldType`

field type for the operation

1.1.21 Struct _CpaCyEcdsaSignRSOpData

- Defined in file_api_lac_cpa_cy_ecdsa.h

Struct Documentation

struct `_CpaCyEcdsaSignRSOpData`

ECDSA Sign R & S Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

cpaCyEcdsaSignRS()

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.

Public Members

CpaFlatBuffer `xg`

x coordinate of base point G

CpaFlatBuffer `yg`

y coordinate of base point G

CpaFlatBuffer `n`

order of the base point G, which shall be prime

CpaFlatBuffer `q`

prime modulus or irreducible polynomial over GF(2^r)

CpaFlatBuffer `a`

a elliptic curve coefficient

CpaFlatBuffer `b`

b elliptic curve coefficient

CpaFlatBuffer `k`

random value (k > 0 and k < n)

CpaFlatBuffer `m`

digest of the message to be signed

CpaFlatBuffer `d`

private key

CpaCyEcFieldType `fieldType`

field type for the operation

1.1.22 Struct _CpaCyEcdsaSignSOpData

- Defined in file_api_lac_cpa_cy_ecdsa.h

Struct Documentation

struct `_CpaCyEcdsaSignSOpData`

ECDSA Sign S Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

cpaCyEcdsaSignS()

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.

Public Members

CpaFlatBuffer `m`

digest of the message to be signed

CpaFlatBuffer `d`

private key

CpaFlatBuffer `r`

Ecdsa r signature value

CpaFlatBuffer `k`

random value (k > 0 and k < n)

CpaFlatBuffer `n`

order of the base point G, which shall be prime

CpaCyEcFieldType `fieldType`

field type for the operation

1.1.23 Struct _CpaCyEcdsaStats64

- Defined in file_api_lac_cpa_cy_ecdsa.h

Struct Documentation

struct `_CpaCyEcdsaStats64`

Cryptographic ECDSA Statistics.

Description: This structure contains statistics on the Cryptographic ECDSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

Public Members

Cpa64U `numEcdsaSignRRequests`

Total number of ECDSA Sign R operation requests.

Cpa64U `numEcdsaSignRRequestErrors`

Total number of ECDSA Sign R operation requests that had an error and could not be processed.

Cpa64U `numEcdsaSignRCompleted`

Total number of ECDSA Sign R operation requests that completed successfully.

Cpa64U `numEcdsaSignRCompletedErrors`

Total number of ECDSA Sign R operation requests that could not be completed successfully due to errors.

Cpa64U `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 `numEcdsaSignSRequests`

Total number of ECDSA Sign S operation requests.

Cpa64U `numEcdsaSignSRequestErrors`

Total number of ECDSA Sign S operation requests that had an error and could not be processed.

Cpa64U `numEcdsaSignSCompleted`

Total number of ECDSA Sign S operation requests that completed successfully.

Cpa64U `numEcdsaSignSCompletedErrors`

Total number of ECDSA Sign S operation requests that could not be completed successfully due to errors.

Cpa64U `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 `numEcdsaSignRSRequests`

Total number of ECDSA Sign R & S operation requests.

Cpa64U `numEcdsaSignRSRequestErrors`

Total number of ECDSA Sign R & S operation requests that had an error and could not be processed.

Cpa64U `numEcdsaSignRSCompleted`

Total number of ECDSA Sign R & S operation requests that completed successfully.

Cpa64U `numEcdsaSignRSCompletedErrors`

Total number of ECDSA Sign R & S operation requests that could not be completed successfully due to errors.

Cpa64U `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 `numEcdsaVerifyRequests`

Total number of ECDSA Verification operation requests.

Cpa64U `numEcdsaVerifyRequestErrors`

Total number of ECDSA Verification operation requests that had an error and could not be processed.

Cpa64U `numEcdsaVerifyCompleted`

Total number of ECDSA Verification operation requests that completed successfully.

Cpa64U `numEcdsaVerifyCompletedErrors`

Total number of ECDSA Verification operation requests that could not be completed successfully due to errors.

Cpa64U `numEcdsaVerifyCompletedOutputInvalid`

Total number of ECDSA Verification operation requests that resulted in an invalid output. Note that this does not indicate an error.

Cpa64U `numKptEcdsaSignRSCompletedOutputInvalid`

Total number of KPT 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 `numKptEcdsaSignRSCompleted`

Total number of KPT ECDSA Sign R & S operation requests that completed successfully.

Cpa64U `numKptEcdsaSignRSRequests`

Total number of KPT ECDSA Sign R & S operation requests.

Cpa64U `numKptEcdsaSignRSRequestErrors`

Total number of KPT ECDSA Sign R & S operation requests that had an error and could not be processed.

Cpa64U `numKptEcdsaSignRSCompletedErrors`

Total number of KPT ECDSA Sign R & S operation requests that could not be completed successfully due to errors.

1.1.24 Struct _CpaCyEcdsaVerifyOpData

- Defined in file_api_lac_cpa_cy_ecdsa.h

Struct Documentation

struct `_CpaCyEcdsaVerifyOpData`

ECDSA Verify Operation Data, for Public Key.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

CpaCyEcdsaVerify()

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.

Public Members

CpaFlatBuffer **xg**

x coordinate of base point G

CpaFlatBuffer **yg**

y coordinate of base point G

CpaFlatBuffer **n**

order of the base point G, which shall be prime

CpaFlatBuffer **q**

prime modulus or irreducible polynomial over GF(2^r)

CpaFlatBuffer **a**

a elliptic curve coefficient

CpaFlatBuffer `b`

b elliptic curve coefficient

CpaFlatBuffer `m`

digest of the message to be signed

CpaFlatBuffer `r`

ECDSA r signature value (r > 0 and r < n)

CpaFlatBuffer `s`

ECDSA s signature value (s > 0 and s < n)

CpaFlatBuffer `xp`

x coordinate of point P (public key)

CpaFlatBuffer `yp`

y coordinate of point P (public key)

CpaCyEcFieldType `fieldType`

field type for the operation

1.1.25 Struct _CpaCyEcGenericPointMultiplyOpData

- Defined in file_api_lac_cpa_cy_ec.h

Struct Documentation

struct `_CpaCyEcGenericPointMultiplyOpData`

Generic EC Point Multiplication Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: This structure contains a generic EC point and a multiplier for use with cpaCyEcGenericPointMultiply. This is common for representing all EC points, irrespective of curve type: Weierstrass, Montgomery and Twisted Edwards (at this time only Weierstrass are supported). The same point + multiplier format can be used when performing generator multiplication, in which case the xP, yP supplied in this structure will be ignored by QAT API library & a generator point will be inserted in their place.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

cpaCyEcGenericPointMultiply()

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

Public Members

CpaFlatBuffer `k`

CpaFlatBuffer `xP`

<scalar multiplier (k > 0 and k < n)

CpaFlatBuffer `yP`

<x coordinate of public key

CpaCyEcCurve *`pCurve`

<y coordinate of public key

CpaBoolean `generator`

<curve type specific parameters

1.1.26 Struct _CpaCyEcGenericPointVerifyOpData

- Defined in file_api_lac_cpa_cy_ec.h

Struct Documentation

struct `_CpaCyEcGenericPointVerifyOpData`

Generic EC Point Verify Operation Data.

This structure contains a generic EC point, irrespective of curve type. It is used to verify when the <x,y> pair specified in the structure lies on the curve indicated in the cpaCyEcGenericPointVerify API.

Description: This structure contains the operation data for the cpaCyEcGenericPointVerify function. This is common for representing all EC points, irrespective of curve type: Weierstrass, Montgomery and Twisted Edwards (at this time only Weierstrass are supported).

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.

See also:

cpaCyEcGenericPointVerify()

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

Public Members

CpaFlatBuffer `xP`

CpaFlatBuffer `yP`

<x coordinate of public key

CpaCyEcCurve *`pCurve`

<y coordinate of public key

1.1.27 Struct _CpaCyEcMontEdwdsPointMultiplyOpData

- Defined in file_api_lac_cpa_cy_ec.h

Struct Documentation

struct `_CpaCyEcMontEdwdsPointMultiplyOpData`

EC Point Multiplication Operation Data for Edwards or Montgomery curves as specificied in RFC#7748.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: This structure contains the operation data for the cpaCyEcMontEdwdsPointMultiply 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. a.pData[0] = MSB.

All buffers in this structure need to be:

- 32 bytes in size for 25519 curves
- 64 bytes in size for 448 curves

See also:

cpaCyEcMontEdwdsPointMultiply()

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

Public Members

CpaCyEcMontEdwdsCurveType `curveType`

field type for the operation

CpaBoolean `generator`

True if the operation is a generator multiplication (kG) False if it is a variable point multiplcation (kP).

CpaFlatBuffer `k`

k scalar multiplier for the operation

CpaFlatBuffer x

x value. Used in scalar varable point multiplication operations. Not required if the generator is True. Must be NULL if not required. The size of the buffer MUST be 32B for 25519 curves and 64B for 448 curves

CpaFlatBuffer y

y value. Used in variable point multiplication of operations. Not required if the generator is True. Must be NULL if not required. The size of the buffer MUST be 32B for 25519 curves and 64B for 448 curves

1.1.28 Struct _CpaCyEcPointMultiplyOpData

- Defined in file_api_lac_cpa_cy_ec.h

Struct Documentation

struct `_CpaCyEcPointMultiplyOpData`

EC Point Multiplication Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

cpaCyEcPointMultiply()

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.

Public Members

CpaFlatBuffer `k`

scalar multiplier (k > 0 and k < n)

CpaFlatBuffer `xg`

x coordinate of curve point

CpaFlatBuffer `yg`

y coordinate of curve point

CpaFlatBuffer `a`

a elliptic curve coefficient

CpaFlatBuffer `b`

b elliptic curve coefficient

CpaFlatBuffer `q`

prime modulus or irreducible polynomial over GF(2^m)

CpaFlatBuffer `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 `fieldType`

field type for the operation

1.1.29 Struct _CpaCyEcPointVerifyOpData

- Defined in file_api_lac_cpa_cy_ec.h

Struct Documentation

struct `_CpaCyEcPointVerifyOpData`

EC Point Verification Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

cpaCyEcPointVerify()

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.

Public Members

CpaFlatBuffer `xq`

x coordinate candidate point

CpaFlatBuffer `yq`

y coordinate candidate point

CpaFlatBuffer `q`

prime modulus or irreducible polynomial over GF(2^m)

CpaFlatBuffer `a`

a elliptic curve coefficient

CpaFlatBuffer `b`

b elliptic curve coefficient

CpaCyEcFieldType `fieldType`

field type for the operation

1.1.30 Struct _CpaCyEcStats64

- Defined in file_api_lac_cpa_cy_ec.h

Struct Documentation

struct `_CpaCyEcStats64`

Cryptographic EC Statistics.

Description: This structure contains statistics on the Cryptographic EC operations. Statistics are set to zero when the component is initialized, and are collected per instance.

Public Members

Cpa64U `numEcPointMultiplyRequests`

Total number of EC Point Multiplication operation requests.

Cpa64U `numEcPointMultiplyRequestErrors`

Total number of EC Point Multiplication operation requests that had an error and could not be processed.

Cpa64U `numEcPointMultiplyCompleted`

Total number of EC Point Multiplication operation requests that completed successfully.

Cpa64U `numEcPointMultiplyCompletedError`

Total number of EC Point Multiplication operation requests that could not be completed successfully due to errors.

Cpa64U `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 `numEcPointVerifyRequests`

Total number of EC Point Verification operation requests.

Cpa64U `numEcPointVerifyRequestErrors`

Total number of EC Point Verification operation requests that had an error and could not be processed.

Cpa64U `numEcPointVerifyCompleted`

Total number of EC Point Verification operation requests that completed successfully.

Cpa64U `numEcPointVerifyCompletedErrors`

Total number of EC Point Verification operation requests that could not be completed successfully due to errors.

Cpa64U `numEcPointVerifyCompletedOutputInvalid`

Total number of EC Point Verification operation requests that had an invalid output. Note that this does not indicate an error.

1.1.31 Struct _CpaCyKeyGenHKDFExpandLabel

- Defined in file_api_lac_cpa_cy_key.h

Struct Documentation

struct `_CpaCyKeyGenHKDFExpandLabel`

Maximum number of labels in op structure

TLS data for key generation functions

File: cpa_cy_key.h

Description: This structure contains data for describing label for the HKDF Extract Label function

Extract Label Function

labelLen = length of the label field

contextLen = length of the context field

sublabelFlag = Mask of sub labels required for this label.

label = label as defined in RFC8446

context = context as defined in RFC8446

Public Members

Cpa8U `label`[CPA_CY_HKDF_KEY_MAX_LABEL_SZ]

HKDFLabel field as defined in RFC8446 sec 7.1.

Cpa8U `labelLen`

The length, in bytes of the label

Cpa8U `sublabelFlag`

mask of sublabels to be generated. This flag is composed of zero or more of: CPA_CY_HKDF_SUBLABEL_KEY CPA_CY_HKDF_SUBLABEL_IV CPA_CY_HKDF_SUBLABEL_RESUMPTION CPA_CY_HKDF_SUBLABEL_FINISHED

1.1.32 Struct _CpaCyKeyGenHKDFOpData

- Defined in file_api_lac_cpa_cy_key.h

Struct Documentation

struct `_CpaCyKeyGenHKDFOpData`

TLS data for key generation functions

Description:

This structure contains data for all HKDF operations:

HKDF Extract

HKDF Expand

HKDF Expand Label

HKDF Extract and Expand

HKDF Extract and Expand Label

HKDF Map Structure Elements

secret - IKM value for extract operations or PRK for expand or expand operations.

seed - contains the salt for extract operations

info - contains the info data for extract operations

labels - See notes above

Public Members

CpaCyKeyHKDFOp `hkdfKeyOp`
: Keying operation to be performed.

Cpa8U `secretLen`
: Length of secret field

Cpa16U `seedLen`
: Length of seed field

Cpa16U `infoLen`
: Length of info field

Cpa16U `numLabels`
: Number of filled CpaCyKeyGenHKDFExpandLabel elements

Cpa8U `secret`[CPA_CY_HKDF_KEY_MAX_SECRET_SZ]
: Input Key Material or PRK

Cpa8U `seed`[CPA_CY_HKDF_KEY_MAX_HMAC_SZ]
: Input salt

Cpa8U `info`[CPA_CY_HKDF_KEY_MAX_INFO_SZ]
: info field

CpaCyKeyGenHKDFExpandLabel `label`[CPA_CY_HKDF_KEY_MAX_LABEL_COUNT]
: array of Expand Label structures

1.1.33 Struct _CpaCyKeyGenMgfOpData

- Defined in file_api_lac_cpa_cy_key.h

Struct Documentation

struct `_CpaCyKeyGenMgfOpData`

Key Generation Mask Generation Function (MGF) Data

See also:

cpaCyKeyGenMgf

Description: 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*.

Public Members

CpaFlatBuffer `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 `maskLenInBytes`

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

1.1.34 Struct _CpaCyKeyGenMgfOpDataExt

- Defined in file_api_lac_cpa_cy_key.h

Struct Documentation

struct `_CpaCyKeyGenMgfOpDataExt`

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

See also:

cpaCyKeyGenMgfExt

Description: 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.

Public Members

CpaCyKeyGenMgfOpData `baseOpData`

"Base" operational data for MGF generation

CpaCySymHashAlgorithm `hashAlgorithm`

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

1.1.35 Struct _CpaCyKeyGenSslOpData

- Defined in file_api_lac_cpa_cy_key.h

Struct Documentation

struct `_CpaCyKeyGenSslOpData`

SSL data for key generation functions

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

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

Note: In each of the above descriptions, + indicates concatenation.

Note: 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.

Public Members

CpaCyKeySslOp `sslOp`

Indicate the SSL operation to be performed

CpaFlatBuffer `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 `seed`

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

CpaFlatBuffer `info`

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

Cpa32U `generatedKeyLenInBytes`

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

CpaFlatBuffer `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, or otherwise it is ignored and can be set to NULL. Implementation-specific limits may apply to this length.

1.1.36 Struct _CpaCyKeyGenStats

- Defined in file_api_lac_cpa_cy_key.h

Struct Documentation

struct `_CpaCyKeyGenStats`

Key Generation Statistics.

Deprecated:

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

Description: 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.

Public Members

Cpa32U `numSslKeyGenRequests`

Total number of successful SSL key generation requests.

Cpa32U `numSslKeyGenRequestErrors`

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

Cpa32U `numSslKeyGenCompleted`

Total number of SSL key generation operations that completed successfully.

Cpa32U `numSslKeyGenCompletedErrors`

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

Cpa32U `numTlsKeyGenRequests`

Total number of successful TLS key generation requests.

Cpa32U `numTlsKeyGenRequestErrors`

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

Cpa32U `numTlsKeyGenCompleted`

Total number of TLS key generation operations that completed successfully.

Cpa32U `numTlsKeyGenCompletedErrors`

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

Cpa32U `numMgfKeyGenRequests`

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

Cpa32U `numMgfKeyGenRequestErrors`

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

Cpa32U `numMgfKeyGenCompleted`

Total number of MGF key generation operations that completed successfully.

Cpa32U `numMgfKeyGenCompletedErrors`

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

1.1.37 Struct _CpaCyKeyGenStats64

- Defined in file_api_lac_cpa_cy_key.h

Struct Documentation

struct `_CpaCyKeyGenStats64`

Key Generation Statistics (64-bit version).

Description: 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.

Public Members

Cpa64U `numSslKeyGenRequests`

Total number of successful SSL key generation requests.

Cpa64U `numSslKeyGenRequestErrors`

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

Cpa64U `numSslKeyGenCompleted`

Total number of SSL key generation operations that completed successfully.

Cpa64U `numSslKeyGenCompletedErrors`

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

Cpa64U `numTlsKeyGenRequests`

Total number of successful TLS key generation requests.

Cpa64U `numTlsKeyGenRequestErrors`

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

Cpa64U `numTlsKeyGenCompleted`

Total number of TLS key generation operations that completed successfully.

Cpa64U `numTlsKeyGenCompletedErrors`

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

Cpa64U `numMgfKeyGenRequests`

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

Cpa64U `numMgfKeyGenRequestErrors`

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

Cpa64U `numMgfKeyGenCompleted`

Total number of MGF key generation operations that completed successfully.

Cpa64U `numMgfKeyGenCompletedErrors`

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

1.1.38 Struct _CpaCyKeyGenTlsOpData

- Defined in file_api_lac_cpa_cy_key.h

Struct Documentation

struct `_CpaCyKeyGenTlsOpData`

TLS data for key generation functions

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

Description: 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

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.

Note: In each of the above descriptions, + indicates concatenation.

Note: 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.

Public Members

CpaCyKeyTlsOp `tlsOp`

TLS operation to be performed

CpaFlatBuffer `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.

CpaFlatBuffer `seed`

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

Cpa32U `generatedKeyLenInBytes`

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

CpaFlatBuffer `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. Implementation-specific limits may apply to this length.

1.1.39 Struct _CpaCyLnModExpOpData

- Defined in file_api_lac_cpa_cy_ln.h

Struct Documentation

struct `_CpaCyLnModExpOpData`

Modular Exponentiation Function Operation Data.

The values of the base, the exponent and the modulus MUST all be less than 2^8192, and the modulus must not be equal to zero.

Description: 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.

Public Members

CpaFlatBuffer `modulus`

Flat buffer containing a pointer to the modulus. This number may be up to 8192 bits in length, and MUST be greater than zero.

CpaFlatBuffer `base`

Flat buffer containing a pointer to the base. This number may be up to 8192 bits in length.

CpaFlatBuffer `exponent`

Flat buffer containing a pointer to the exponent. This number may be up to 8192 bits in length.

1.1.40 Struct _CpaCyLnModInvOpData

- Defined in file_api_lac_cpa_cy_ln.h

Struct Documentation

struct `_CpaCyLnModInvOpData`

Modular Inversion Function Operation Data.

Note that the values of A and B MUST NOT both be even numbers, and both MUST be less than 2^8192.

Description: 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.

Public Members

CpaFlatBuffer `A`

Flat buffer containing a pointer to the value that will be inverted. This number may be up to 8192 bits in length, it MUST NOT be zero, and it MUST be co-prime with B.

CpaFlatBuffer `B`

Flat buffer containing a pointer to the value that will be used as the modulus. This number may be up to 8192 bits in length, it MUST NOT be zero, and it MUST be co-prime with A.

1.1.41 Struct _CpaCyLnStats

- Defined in file_api_lac_cpa_cy_ln.h

Struct Documentation

struct `_CpaCyLnStats`

Look Aside Cryptographic large number Statistics.

Deprecated:

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

Description: 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.

Public Members

Cpa32U `numLnModExpRequests`

Total number of successful large number modular exponentiation requests.

Cpa32U `numLnModExpRequestErrors`

Total number of large number modular exponentiation requests that had an error and could not be processed.

Cpa32U `numLnModExpCompleted`

Total number of large number modular exponentiation operations that completed successfully.

Cpa32U `numLnModExpCompletedErrors`

Total number of large number modular exponentiation operations that could not be completed successfully due to errors.

Cpa32U `numLnModInvRequests`

Total number of successful large number modular inversion requests.

Cpa32U `numLnModInvRequestErrors`

Total number of large number modular inversion requests that had an error and could not be processed.

Cpa32U `numLnModInvCompleted`

Total number of large number modular inversion operations that completed successfully.

Cpa32U `numLnModInvCompletedErrors`

Total number of large number modular inversion operations that could not be completed successfully due to errors.

1.1.42 Struct _CpaCyLnStats64

- Defined in file_api_lac_cpa_cy_ln.h

Struct Documentation

struct `_CpaCyLnStats64`

Look Aside Cryptographic large number Statistics.

Description: 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.

Public Members

Cpa64U `numLnModExpRequests`

Total number of successful large number modular exponentiation requests.

Cpa64U `numLnModExpRequestErrors`

Total number of large number modular exponentiation requests that had an error and could not be processed.

Cpa64U `numLnModExpCompleted`

Total number of large number modular exponentiation operations that completed successfully.

Cpa64U `numLnModExpCompletedErrors`

Total number of large number modular exponentiation operations that could not be completed successfully due to errors.

Cpa64U `numLnModInvRequests`

Total number of successful large number modular inversion requests.

Cpa64U `numLnModInvRequestErrors`

Total number of large number modular inversion requests that had an error and could not be processed.

Cpa64U `numLnModInvCompleted`

Total number of large number modular inversion operations that completed successfully.

Cpa64U `numLnModInvCompletedErrors`

Total number of large number modular inversion operations that could not be completed successfully due to errors.

1.1.43 Struct _CpaCyPrimeStats

- Defined in file_api_lac_cpa_cy_prime.h

Struct Documentation

struct `_CpaCyPrimeStats`

Prime Number Test Statistics.

Deprecated:

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

Description: 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.

Public Members

Cpa32U `numPrimeTestRequests`

Total number of successful prime number test requests.

Cpa32U `numPrimeTestRequestErrors`

Total number of prime number test requests that had an error and could not be processed.

Cpa32U `numPrimeTestCompleted`

Total number of prime number test operations that completed successfully.

Cpa32U `numPrimeTestCompletedErrors`

Total number of prime number test operations that could not be completed successfully due to errors.

Cpa32U `numPrimeTestFailures`

Total number of prime number test operations that executed successfully but the outcome of the test was that the number was not prime.

1.1.44 Struct _CpaCyPrimeStats64

- Defined in file_api_lac_cpa_cy_prime.h

Struct Documentation

struct `_CpaCyPrimeStats64`

Prime Number Test Statistics (64-bit version).

Description: 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.

Public Members

Cpa64U `numPrimeTestRequests`

Total number of successful prime number test requests.

Cpa64U `numPrimeTestRequestErrors`

Total number of prime number test requests that had an error and could not be processed.

Cpa64U `numPrimeTestCompleted`

Total number of prime number test operations that completed successfully.

Cpa64U `numPrimeTestCompletedErrors`

Total number of prime number test operations that could not be completed successfully due to errors.

Cpa64U `numPrimeTestFailures`

Total number of prime number test operations that executed successfully but the outcome of the test was that the number was not prime.

1.1.45 Struct _CpaCyPrimeTestOpData

- Defined in file_api_lac_cpa_cy_prime.h

Struct Documentation

struct `_CpaCyPrimeTestOpData`

Prime Test Operation Data.

All values in this structure are required to be in Most Significant Byte first order, e.g. primeCandidate.pData[0] = MSB.

Description: 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 numbers MUST be stored in big-endian order.

See also:

cpaCyPrimeTest()

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.

Public Members

CpaFlatBuffer `primeCandidate`

The prime number candidate to test

CpaBoolean `performGcdTest`

A value of CPA_TRUE means perform a GCD Primality Test

CpaBoolean `performFermatTest`

A value of CPA_TRUE means perform a Fermat Primality Test

Cpa32U `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 `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 `performLucasTest`

An CPA_TRUE value means perform a Lucas Primality Test

1.1.46 Struct _CpaCyRsaDecryptOpData

- Defined in file_api_lac_cpa_cy_rsa.h

Struct Documentation

struct `_CpaCyRsaDecryptOpData`

RSA Decryption Primitive Operation Data

Description: 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.

Public Members

CpaCyRsaPrivateKey `*pRecipientPrivateKey`

Pointer to the recipient's RSA private key.

CpaFlatBuffer `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.

1.1.47 Struct _CpaCyRsaEncryptOpData

- Defined in file_api_lac_cpa_cy_rsa.h

Struct Documentation

struct `_CpaCyRsaEncryptOpData`

RSA Encryption Primitive Operation Data

Description: 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.

Public Members

CpaCyRsaPublicKey *`pPublicKey`

Pointer to the public key.

CpaFlatBuffer `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.

1.1.48 Struct _CpaCyRsaKeyGenOpData

- Defined in file_api_lac_cpa_cy_rsa.h

Struct Documentation

struct `_CpaCyRsaKeyGenOpData`

RSA Key Generation Data.

The following limitations on the permutations of the supported bit lengths of p, q and n (written as {p, q, n}) apply:

Description: 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.

- {256, 256, 512} or
- {512, 512, 1024} or
- {768, 768, 1536} or
- {1024, 1024, 2048} or
- {1536, 1536, 3072} or
- {2048, 2048, 4096}.

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.

Public Members

CpaFlatBuffer `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 `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 `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 `version`

Indicates the version of the PKCS #1 specification that is supported. Note that this applies to both representations.

CpaCyRsaPrivateKeyRepType `privateKeyRepType`

This value is used to identify which of the private key representation types is required to be generated.

CpaFlatBuffer `publicExponentE`

The public exponent (e).

1.1.49 Struct _CpaCyRsaPrivateKey

- Defined in file_api_lac_cpa_cy_rsa.h

Struct Documentation

struct `_CpaCyRsaPrivateKey`

RSA Private Key Structure.

Description: 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.

Public Members

CpaCyRsaVersion `version`

Indicates the version of the PKCS #1 specification that is supported. Note that this applies to both representations.

CpaCyRsaPrivateKeyRepType `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 `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 `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.

1.1.50 Struct _CpaCyRsaPrivateKeyRep1

- Defined in file_api_lac_cpa_cy_rsa.h

Struct Documentation

struct `_CpaCyRsaPrivateKeyRep1`

RSA Private Key Structure For Representation 1.

Description: 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.

Public Members

CpaFlatBuffer `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),
- 4096 bits (512 bytes), or
- 8192 bits (1024 bytes).

CpaFlatBuffer `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.

1.1.51 Struct _CpaCyRsaPrivateKeyRep2

- Defined in file_api_lac_cpa_cy_rsa.h

Struct Documentation

struct `_CpaCyRsaPrivateKeyRep2`

RSA Private Key Structure For Representation 2.

Description: 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.

Public Members

CpaFlatBuffer `prime1P`

The first large prime (p). For key generation operations, this field is unused.

CpaFlatBuffer `prime2Q`

The second large prime (q). For key generation operations, this field is unused.

CpaFlatBuffer `exponent1Dp`

The first factor CRT exponent (dP). d mod (p-1).

CpaFlatBuffer `exponent2Dq`

The second factor CRT exponent (dQ). d mod (q-1).

CpaFlatBuffer `coefficientQInv`

The (first) Chinese Remainder Theorem (CRT) coefficient (qInv). (inverse of q) mod p.

1.1.52 Struct _CpaCyRsaPublicKey

- Defined in file_api_lac_cpa_cy_rsa.h

Struct Documentation

struct `_CpaCyRsaPublicKey`

RSA Public Key Structure.

Description: 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.

Public Members

CpaFlatBuffer `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 `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.

1.1.53 Struct _CpaCyRsaStats

- Defined in file_api_lac_cpa_cy_rsa.h

Struct Documentation

struct `_CpaCyRsaStats`

RSA Statistics.

Deprecated:

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

Description: This structure contains statistics on the RSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

Public Members

Cpa32U `numRsaKeyGenRequests`

Total number of successful RSA key generation requests.

Cpa32U `numRsaKeyGenRequestErrors`

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

Cpa32U `numRsaKeyGenCompleted`

Total number of RSA key generation operations that completed successfully.

Cpa32U `numRsaKeyGenCompletedErrors`

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

Cpa32U `numRsaEncryptRequests`

Total number of successful RSA encrypt operation requests.

Cpa32U `numRsaEncryptRequestErrors`

Total number of RSA encrypt requests that had an error and could not be processed.

Cpa32U `numRsaEncryptCompleted`

Total number of RSA encrypt operations that completed successfully.

Cpa32U `numRsaEncryptCompletedErrors`

Total number of RSA encrypt operations that could not be completed successfully due to errors.

Cpa32U `numRsaDecryptRequests`

Total number of successful RSA decrypt operation requests.

Cpa32U `numRsaDecryptRequestErrors`

Total number of RSA decrypt requests that had an error and could not be processed.

Cpa32U `numRsaDecryptCompleted`

Total number of RSA decrypt operations that completed successfully.

Cpa32U `numRsaDecryptCompletedErrors`

Total number of RSA decrypt operations that could not be completed successfully due to errors.

1.1.54 Struct _CpaCyRsaStats64

- Defined in file_api_lac_cpa_cy_rsa.h

Struct Documentation

struct `_CpaCyRsaStats64`

RSA Statistics (64-bit version).

Description: 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.

Public Members

Cpa64U `numRsaKeyGenRequests`

Total number of successful RSA key generation requests.

Cpa64U `numRsaKeyGenRequestErrors`

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

Cpa64U `numRsaKeyGenCompleted`

Total number of RSA key generation operations that completed successfully.

Cpa64U `numRsaKeyGenCompletedErrors`

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

Cpa64U `numRsaEncryptRequests`

Total number of successful RSA encrypt operation requests.

Cpa64U `numRsaEncryptRequestErrors`

Total number of RSA encrypt requests that had an error and could not be processed.

Cpa64U `numRsaEncryptCompleted`

Total number of RSA encrypt operations that completed successfully.

Cpa64U `numRsaEncryptCompletedErrors`

Total number of RSA encrypt operations that could not be completed successfully due to errors.

Cpa64U `numRsaDecryptRequests`

Total number of successful RSA decrypt operation requests.

Cpa64U `numRsaDecryptRequestErrors`

Total number of RSA decrypt requests that had an error and could not be processed.

Cpa64U `numRsaDecryptCompleted`

Total number of RSA decrypt operations that completed successfully.

Cpa64U `numRsaDecryptCompletedErrors`

Total number of RSA decrypt operations that could not be completed successfully due to errors.

Cpa64U `numKptRsaDecryptRequests`

Total number of successful KPT RSA decrypt operation requests.

Cpa64U `numKptRsaDecryptRequestErrors`

Total number of KPT RSA decrypt requests that had an error and could not be processed.

Cpa64U `numKptRsaDecryptCompleted`

Total number of KPT RSA decrypt operations that completed successfully.

Cpa64U `numKptRsaDecryptCompletedErrors`

Total number of KPT RSA decrypt operations that could not be completed successfully due to errors.

1.1.55 Struct _CpaCySymCapabilitiesInfo

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymCapabilitiesInfo`

Symmetric Capabilities Info

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

Description: 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.

```
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.
```

Public Functions

`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*.

`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*.

Public Members

CpaBoolean `partialPacketSupported`

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

1.1.56 Struct _CpaCySymCipherSetupData

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymCipherSetupData`

Symmetric Cipher Setup Data.

Description: This structure contains data relating to Cipher (Encryption and Decryption) to setup a session.

Public Members

CpaCySymCipherAlgorithm `cipherAlgorithm`

Cipher algorithm and mode

Cpa32U `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 key-mask 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 *`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. These fields are set to NULL if key derivation will be used.

CpaCySymCipherDirection `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.

1.1.57 Struct _CpaCySymDeriveOpData

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymDeriveOpData`

Symmetric Cipher Op Data for key derivation

File: cpa_cy_sym.h

Description: This structure contains the cipher key or the data to derive the cipher key in addition to other cipher related data.

Public Members

Cpa8U *`pContext`

Pointer to Context structure

Cpa16U `contextLen`

The number of octets of context

1.1.58 Struct _CpaCySymDpOpData

- Defined in file_api_lac_cpa_cy_sym_dp.h

Struct Documentation

struct `_CpaCySymDpOpData`

Operation Data for cryptographic data plane API.

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

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

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

See also:

cpaCySymDpEnqueueOp, cpaCySymDpEnqueueOpBatch

Public Members

Cpa64U `reserved0`

Reserved for internal usage.

Cpa32U `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 `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.

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

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

CpaPhysicalAddr `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 `reserved1`

Reserved for internal usage.

Cpa32U `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.

Note: 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 `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.

Note: 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.

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

CpaPhysicalAddr `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 `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.

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.

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

CpaInstanceHandle `instanceHandle`

Instance to which the request is to be enqueued.

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

CpaCySymDpSessionCtx `sessionCtx`

Session context specifying the cryptographic parameters for this request.

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

Cpa32U `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 `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 `srcBufferLen`

Length of source buffer, or *CPA_DP_BUFLIST*.

CpaPhysicalAddr `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 `dstBufferLen`

Length of destination buffer, or *CPA_DP_BUFLIST*.

CpaPhysicalAddr `thisPhys`

Physical address of this data structure

Cpa8U *`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 *`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 *`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.

1.1.59 Struct _CpaCySymHashAuthModeSetupData

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymHashAuthModeSetupData`

Hash Auth Mode Setup Data.

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

Public Members

Cpa8U *`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 `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 `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.

1.1.60 Struct _CpaCySymHashNestedModeSetupData

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymHashNestedModeSetupData`

Hash Mode Nested Setup Data.

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

Public Members

Cpa8U *`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 `innerPrefixLenInBytes`

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

CpaCySymHashAlgorithm `outerHashAlgorithm`

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

Cpa8U *`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 `outerPrefixLenInBytes`

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

1.1.61 Struct _CpaCySymHashSetupData

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymHashSetupData`

Hash Setup Data.

Description: 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.

Public Members

CpaCySymHashAlgorithm `hashAlgorithm`

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

CpaCySymHashMode `hashMode`

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

Cpa32U `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 `authModeSetupData`

Authentication Mode Setup Data. Only valid for mode CPA_CY_SYM_MODE_HASH_AUTH

CpaCySymHashNestedModeSetupData `nestedModeSetupData`

Nested Hash Mode Setup Data Only valid for mode CPA_CY_SYM_MODE_HASH_NESTED

1.1.62 Struct _CpaCySymOpData

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymOpData`

Cryptographic Component Operation Data.

See also:

CpaCySymPacketType

Description: 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.

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.

Public Members

CpaCySymSessionCtx `sessionCtx`

Handle for the initialized session context

CpaCySymPacketType `packetType`

Selects the packet type

Cpa8U `*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 `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 `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 `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.

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

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

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

Cpa32U `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.

Note: 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 `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.

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

Note: 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. The maximum length supported for AAD data for AES-GMAC is 16383 bytes.

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

Cpa8U `*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.

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.

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

Cpa8U *`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.

1.1.63 Struct _CpaCySymOpData2

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymOpData2`

Cryptographic Component Operation Data with additional arguments

If the deriveCtxData structure contains non-NULL entries for the context structure, this indicates the cipher key and initialization vector will be either supplied in, or derived from, that context structure. In this case, the pointers to and the lengths of the cipher key and iv in the symOpData structure must be NULL and zero respectively.

Description: 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. It includes a structure to support a NIST 800-108 key derivation. This data structure is currently only used in chained usecases.

Additionally, if the cipher key is provided in the symOpData then the deriveCtxData fields must be set to NULL and zero.

See also:

CpaCySymPacketType

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

Public Members

CpaCySymOpData `symOpData`

Symetric opdata.

CpaCySymDeriveOpData `deriveCtxData`

Key derivation specifc opdata.

1.1.64 Struct _CpaCySymSessionSetupData

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymSessionSetupData`

Session Setup Data.

Description: 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.

Public Members

CpaCyPriority `sessionPriority`

Priority of this session

CpaCySymOp `symOperation`

Operation to perfom

CpaCySymCipherSetupData `cipherSetupData`

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

CpaCySymHashSetupData `hashSetupData`

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

CpaCySymAlgChainOrder `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 `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.

Note: 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 `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.

Note: 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 `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.

1.1.65 Struct _CpaCySymSessionSetupData2

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymSessionSetupData2`

Session setup data with additional Crc Control Data

Description: This structure contains data relating to setting up a session and the crc control data used for data integrity computations. The client needs to complete the information in this structure in order to setup a session.

Public Members

CpaCySymSessionSetupData `symSessionSetupData`

Symmetric crypto session setup data.

CpaCrcControlData `cySessionCrcControlData`

The Crc control data used for this session's data integrity computations

1.1.66 Struct _CpaCySymSessionUpdateData

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymSessionUpdateData`

Session Update Data.

Description: This structure contains data relating to resetting a session.

Public Members

Cpa32U `flags`

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

Cpa8U `*pCipherKey`

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

CpaCySymCipherDirection `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 `*authKey`

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

1.1.67 Struct _CpaCySymStats

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymStats`

Cryptographic Component Statistics.

Deprecated:

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

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

Public Members

Cpa32U `numSessionsInitialized`

Number of session initialized

Cpa32U `numSessionsRemoved`

Number of sessions removed

Cpa32U `numSessionErrors`

Number of session initialized and removed errors.

Cpa32U `numSymOpRequests`

Number of successful symmetric operation requests.

Cpa32U `numSymOpRequestErrors`

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

Cpa32U `numSymOpCompleted`

Number of operations that completed successfully.

Cpa32U `numSymOpCompletedErrors`

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

Cpa32U `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.

1.1.68 Struct _CpaCySymStats64

- Defined in file_api_lac_cpa_cy_sym.h

Struct Documentation

struct `_CpaCySymStats64`

Cryptographic Component Statistics (64-bit version).

Description: 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.

Public Members

Cpa64U `numSessionsInitialized`
Number of session initialized

Cpa64U `numSessionsRemoved`
Number of sessions removed

Cpa64U `numSessionErrors`
Number of session initialized and removed errors.

Cpa64U `numSymOpRequests`
Number of successful symmetric operation requests.

Cpa64U `numSymOpRequestErrors`
Number of operation requests that had an error and could not be processed.

Cpa64U `numSymOpCompleted`
Number of operations that completed successfully.

Cpa64U `numSymOpCompletedErrors`
Number of operations that could not be completed successfully due to errors.

Cpa64U `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.

1.1.69 Struct _CpaDcChainOpData

- Defined in file_api_dc_cpa_dc_chain.h

Struct Documentation

struct `_CpaDcChainOpData`
Compression chaining request input parameters.

Description: This structure contains the request information to use with compression chaining operations.

Public Members

CpaDcChainSessionType `opType`

Indicate the type for this operation

CpaDcOpData *`pDcOp`

Pointer to compression operation data

CpaCySymOpData *`pCySymOp`

Pointer to symmetric crypto operation data with append crc64 option

union *_CpaDcChainOpData*::[anonymous] `[anonymous]`

1.1.70 Struct _CpaDcChainOpData2

- Defined in file_api_dc_cpa_dc_chain.h

Struct Documentation

struct `_CpaDcChainOpData2`

Chaining request op2 data for chained operations with optional verification step.

Description: This structure contains arguments for the cpaDcChainPerformOp2 API.

Public Members

CpaBoolean `testIntegrity`

True if integrity check is required

CpaDcChainOperations `operation`

Description of operations

Cpa8U `numOpDatas`

number of elements in pChainOpData, which define the chain op

CpaDcChainSubOpData2 *`pChainOpData`

array of operations for chaining

1.1.71 Struct _CpaDcChainRqResults

- Defined in file_api_dc_cpa_dc_chain.h

Struct Documentation

struct `_CpaDcChainRqResults`

Chaining request results data

Description: This structure contains the request results.

Public Members

CpaDcReqStatus `dcStatus`

Additional status details from compression accelerator

CpaStatus `cyStatus`

Additional status details from symmetric crypto accelerator

CpaBoolean `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

Cpa32U `produced`

Octets produced to the output buffer

Cpa32U `consumed`

Octets consumed from the input buffer

Cpa32U `crc32`

crc32 checksum produced by chaining operations

Cpa32U `adler32`

Adler32 checksum produced by chaining operations

1.1.72 Struct _CpaDcChainRqVResults

- Defined in file_api_dc_cpa_dc_chain.h

Struct Documentation

struct `_CpaDcChainRqVResults`

Chaining request results data for chained operations with optional verification step.

Description: This structure contains the request results.

Public Members

CpaDcChainRqResults `chainRqResults`

Chain results structure.

Cpa64U `ctxCrc64`

crc64 of context structure applicable to store2 flow

Cpa64U `iDcCrc64`

input crc of DC operation

Cpa64U `oDcCrc64`

output crc of DC operation

Cpa64U `storedCrc64`

crc64 that was appended to compressed data. Only valid if decrypt and decompress operations were performed successfully and appendCrc was requested.

CpaBoolean `reserved1`

Reserved for future use

CpaStatus `chainStatus`

Status of chain command. In the event the request is invalid, a value of CPA_STATUS_INVALID_PARAM or CPA_STATUS_UNSUPPORTED may be returned. If the request has been successfully sent to the accelerator and has been processed, this value will be set to CPA_STATUS_SUCCESS. To verify the success of the entire chain operation, the return status of each operation must be examined.

1.1.73 Struct _CpaDcChainSessionSetupData

- Defined in file_api_dc_cpa_dc_chain.h

Struct Documentation

struct `_CpaDcChainSessionSetupData`

Chaining Session Setup Data.

Description: This structure contains data relating to set up chaining sessions. The client needs to complete the information in this structure in order to setup chaining sessions.

Public Members

CpaDcChainSessionType `sessType`

CpaDcSessionSetupData *`pDcSetupData`

Pointer to compression session setup data

CpaCySymSessionSetupData *`pCySetupData`

Pointer to symmetric crypto session setup data

union *_CpaDcChainSessionSetupData*::[anonymous] `[anonymous]`

Indicate the type for this session

1.1.74 Struct _CpaDcChainSessionSetupData2

- Defined in file_api_dc_cpa_dc_chain.h

Struct Documentation

struct `_CpaDcChainSessionSetupData2`

Chaining session setup data with extended version of Dc and Cy Session setup data structure. Applications which wants to use the CpaDcSessionSetupData2 and CpaCySymSessionSetupData2 structures.

Description: This structure contains data relating to set up chaining sessions. The client needs to complete the information in this structure in order to setup chaining sessions.

Public Members

CpaDcChainSessionType `sessType`

CpaDcSessionSetupData2 *`pDcSetupData`

Pointer to compression session setup data

CpaCySymSessionSetupData2 *`pCySetupData`

Pointer to symmetric crypto session setup data

union *_CpaDcChainSessionSetupData2*::[anonymous] `[anonymous]`

Indicate the type for this session

1.1.75 Struct _CpaDcChainSubOpData2

- Defined in file_api_dc_cpa_dc_chain.h

Struct Documentation

struct `_CpaDcChainSubOpData2`

Compression chaining request input parameters.

Description: This structure contains the request information to use with compression chaining sub-request operations.

Public Members

CpaDcChainSessionType `opType`

Indicate the type for this operation

CpaDcOpData2 *`pDcOp2`

Pointer to compression operation data

CpaCySymOpData2 *`pCySymOp2`

Pointer to symmetric crypto operation data with additional options.

union *_CpaDcChainSubOpData2*::[anonymous] `[anonymous]`

1.1.76 Struct _CpaDcDpOpData

- Defined in file_api_dc_cpa_dc_dp.h

Struct Documentation

struct `_CpaDcDpOpData`

Operation Data for compression data plane API.

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

Description: This structure contains data relating to a request to perform compression processing on one or more data buffers.

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

See also:

cpaDcDpEnqueueOp, cpaDcDpEnqueueOpBatch

Public Members

Cpa64U `reserved0`

Reserved for internal use. Source code should not read or write this field.

Cpa32U `bufferLenToCompress`

The number of bytes from the source buffer to compress. This must be less than, or more typically equal to, the total size of the source buffer (or buffer list).

Cpa32U `bufferLenForData`

The maximum number of bytes that should be written to the destination buffer. This must be less than, or more typically equal to, the total size of the destination buffer (or buffer list).

Cpa64U `reserved1`

Reserved for internal use. Source code should not read or write

Cpa64U `reserved2`

Reserved for internal use. Source code should not read or write

Cpa64U `reserved3`

Reserved for internal use. Source code should not read or write

CpaDcRqResults `results`

Results of the operation. Contents are valid upon completion.

CpaInstanceHandle `dcInstance`

Instance to which the request is to be enqueued

CpaDcSessionHandle `pSessionHandle`

DC Session associated with the stream of requests. This field is only valid when using the session based API functions. This field must be set to NULL if the application wishes to use the No-Session (Ns) API.

CpaPhysicalAddr `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.

Cpa32U `srcBufferLen`

If the source buffer is a "flat buffer", then this field specifies the size of the buffer, in bytes. If the source buffer is a "buffer list" (of type *CpaPhysBufferList*), then this field should be set to the value *CPA_DP_BUFLIST*.

CpaPhysicalAddr `destBuffer`

Physical address of the destination buffer on which to operate. This is either the location of the data, of length destBufferLen; or, if destBufferLen has the special value *CPA_DP_BUFLIST*, then destBuffer contains the location where a *CpaPhysBufferList* is stored.

Cpa32U `destBufferLen`

If the destination buffer is a "flat buffer", then this field specifies the size of the buffer, in bytes. If the destination buffer is a "buffer list" (of type *CpaPhysBufferList*), then this field should be set to the value *CPA_DP_BUFLIST*.

CpaDcSessionDir `sessDirection`

Session direction indicating whether session is used for compression, decompression. For the DP implementation, CPA_DC_DIR_COMBINED is not a valid selection.

CpaBoolean `compressAndVerify`

If set to true, for compression operations, the implementation will verify that compressed data, generated by the compression operation, can be successfully decompressed. This behavior is only supported for stateless compression. This behavior is only supported on instances that support the compressAndVerify capability.

CpaBoolean `compressAndVerifyAndRecover`

If set to true, for compression operations, the implementation will automatically recover from a compressAndVerify error. This behavior is only supported for stateless compression. This behavior is only supported on instances that support the compressAndVerifyAndRecover capability. The compressAndVerify field in CpaDcOpData MUST be set to CPA_TRUE if compressAndVerifyAndRecover is set to CPA_TRUE.

CpaStatus `responseStatus`

Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

CpaPhysicalAddr `thisPhys`

Physical address of this data structure

void *`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.

CpaDcNsSetupData *`pSetupData`

Pointer to the No-session (Ns) Setup data for configuration of this request.

This CpaDcNsSetupData structure must be initialised when using the Data Plane No-Session (Ns) API. Otherwise it should be set to NULL. When initialized, the existing Data Plane API functions can be used as is.

1.1.77 Struct _CpaDcInstanceCapabilities

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaDcInstanceCapabilities`

Implementation Capabilities Structure

Description: This structure contains data relating to the capabilities of an implementation. The capabilities include supported compression algorithms, RFC 1951 options and whether the implementation supports both stateful and stateless compress and decompress sessions.

Public Functions

`CPA_BITMAP`(dcChainCapInfo, CPA_DC_CHAIN_CAP_BITMAP_SIZE)

Bitmap representing which chaining capabilities 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 *CpaDcChainOperations* in cpa_dc_chain.h.

Public Members

CpaBoolean `statefulLZSCompression`

True if the Instance supports Stateful LZS compression

CpaBoolean `statefulLZSDecompression`

True if the Instance supports Stateful LZS decompression

CpaBoolean `statelessLZSCompression`

True if the Instance supports Stateless LZS compression

CpaBoolean `statelessLZSDecompression`

True if the Instance supports Stateless LZS decompression

CpaBoolean `statefulLZSSCompression`

True if the Instance supports Stateful LZSS compression

CpaBoolean `statefulLZSSDecompression`

True if the Instance supports Stateful LZSS decompression

CpaBoolean `statelessLZSSCompression`

True if the Instance supports Stateless LZSS compression

CpaBoolean `statelessLZSSDecompression`

True if the Instance supports Stateless LZSS decompression

CpaBoolean `statefulELZSCompression`

True if the Instance supports Stateful Extended LZS compression

CpaBoolean `statefulELZSDecompression`

True if the Instance supports Stateful Extended LZS decompression

CpaBoolean `statelessELZSCompression`

True if the Instance supports Stateless Extended LZS compression

CpaBoolean `statelessELZSDecompression`

True if the Instance supports Stateless Extended LZS decompression

CpaBoolean `statefulDeflateCompression`

True if the Instance supports Stateful Deflate compression

CpaBoolean `statefulDeflateDecompression`

True if the Instance supports Stateful Deflate decompression

CpaBoolean `statelessDeflateCompression`

True if the Instance supports Stateless Deflate compression

CpaBoolean `statelessDeflateDecompression`

True if the Instance supports Stateless Deflate decompression

CpaBoolean `statelessLZ4Compression`

True if the Instance supports Stateless LZ4 compression

CpaBoolean `statelessLZ4Decompression`

True if the Instance supports Stateless LZ4 decompression

CpaBoolean `statefulLZ4Decompression`

True if the Instance supports Stateful LZ4 decompression

CpaBoolean `statelessLZ4SCompression`

True if the Instance supports Stateless LZ4S compression

CpaBoolean `checksumCRC32`

True if the Instance can calculate a CRC32 checksum over the uncompressed data. This value is only calculated when CPA_DC_DEFLATE is configured as the algorithm for CpaDcCompType

CpaBoolean `checksumAdler32`

True if the Instance can calculate an Adler-32 checksum over the uncompressed data. This value is only calculated when CPA_DC_DEFLATE is configured as the algorithm for CpaDcCompType

CpaBoolean `checksumXXHash32`

True if the Instance can calculate an xxHash-32 hash over the uncompressed data. This value is only calculated when CPA_DC_LZ4 or CPA_DC_LZ4S is configured as the algorithm for CpaDcCompType

CpaBoolean `dynamicHuffman`

True if the Instance supports dynamic Huffman trees in deflate blocks

CpaBoolean `dynamicHuffmanBufferReq`

True if an Instance specific buffer is required to perform a dynamic Huffman tree deflate request

CpaBoolean `precompiledHuffman`

True if the Instance supports precompiled Huffman trees in deflate blocks

CpaBoolean `autoSelectBestHuffmanTree`

True if the Instance has the ability to automatically select between different Huffman encoding schemes for better compression ratios

Cpa8U `validWindowSizeMaskCompression`

Bits set to '1' for each valid window size supported by the compression implementation

Cpa8U `validWindowSizeMaskDecompression`

Bits set to '1' for each valid window size supported by the decompression implementation

Cpa32U `internalHuffmanMem`

Number of bytes internally available to be used when constructing dynamic Huffman trees.

CpaBoolean `endOfLastBlock`

True if the Instance supports stopping at the end of the last block in a deflate stream during a decompression operation and reporting that the end of the last block has been reached as part of the CpaDcReqStatus data.

CpaBoolean `reportParityError`

True if the instance supports parity error reporting.

CpaBoolean `batchAndPack`

True if the instance supports 'batch and pack' compression

CpaBoolean `compressAndVerify`

True if the instance supports checking that compressed data, generated as part of a compression operation, can be successfully decompressed.

CpaBoolean `compressAndVerifyStrict`

True if compressAndVerify is 'strictly' enabled for the instance. If strictly enabled, compressAndVerify will be enabled by default for compression operations and cannot be disabled by setting opData.compressAndVerify=0 with *cpaDcCompressData2()*. Compression operations with opData.compressAndVerify=0 will return a CPA_STATUS_INVALID_PARAM error status when in compressAndVerify strict mode.

CpaBoolean `compressAndVerifyAndRecover`

True if the instance supports recovering from errors detected by compressAndVerify by generating a stored block in the compressed output data buffer. This stored block replaces any compressed content that resulted in a compressAndVerify error.

CpaBoolean `integrityCrcs`

True if the instance supports 32 bit integrity CRC checking in the compression/decompression datapath. Refer to *CpaDcOpData* for more details on integrity checking.

CpaBoolean `integrityCrcs64b`

True if the instance supports 64 bit integrity CRC checking in the compression / decompression datapath. Refer to *CpaDcOpData* for more details on integrity checking.

1.1.78 Struct _CpaDcOpData

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaDcOpData`

(De)Compression request input parameters.

Description: This structure contains the request information for use with compression operations.

Public Members

CpaDcFlush `flushFlag`

Indicates the type of flush to be performed.

CpaBoolean `compressAndVerify`

If set to true, for compression operations, the implementation will verify that compressed data, generated by the compression operation, can be successfully decompressed. This behavior is only supported for stateless compression. This behavior is only supported on instances that support the compressAndVerify capability.

CpaBoolean `compressAndVerifyAndRecover`

If set to true, for compression operations, the implementation will automatically recover from a compressAndVerify error. This behavior is only supported for stateless compression. This behavior is only supported on instances that support the compressAndVerifyAndRecover capability. The compressAndVerify field in CpaDcOpData MUST be set to CPA_TRUE if compressAndVerifyAndRecover is set to CPA_TRUE.

CpaBoolean `integrityCrcCheck`

If set to true, the implementation will verify that data integrity is preserved through the processing pipeline.

Integrity CRC checking is not supported for decompression operations over data that contains multiple gzip headers.

CpaBoolean `verifyHwIntegrityCrcs`

If set to true, software calculated CRCs will be compared against hardware generated integrity CRCs to ensure that data integrity is maintained when transferring data to and from the hardware accelerator.

CpaDcIntegrityCrcSize `integrityCrcSize`

This option specifies the size of the CRC to be used for data integrity checking. As such it is only valid if this request is configured for data integrity checks.

CpaDcSkipData `inputSkipData`

Optional skip regions in the input buffers

CpaDcSkipData `outputSkipData`

Optional skip regions in the output buffers

CpaCrcData *`pCrcData`

Pointer to CRCs for this operation, when integrity checks are enabled.

1.1.79 Struct _CpaDcOpData2

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaDcOpData2`

(De)Compression request input parameters.

Description: This structure contains the request information for use with compression operations with additional options

Public Members

CpaDcOpData `dcOpData`

Data for compress operation.

CpaBoolean `appendCRC64`

1.1.80 Struct _CpaDcRqResults

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaDcRqResults`

Request results data

For stateful sessions the status, produced, consumed and endOfLastBlock results are per request values while the checksum value is cumulative across all requests on the session so far. In this case the checksum value is not guaranteed to be correct until the final compressed data has been processed.

Description: This structure contains the request results.

For stateless sessions, an initial checksum value is passed into the stateless operation. Once the stateless operation completes, the checksum value will contain checksum produced by the operation.

Public Members

CpaDcReqStatus `status`

Additional status details from accelerator

Cpa32U `produced`

Octets produced by the operation

Cpa32U `consumed`

Octets consumed by the operation

Cpa32U `checksum`

The checksum produced by the operation. For some checksum algorithms, setting this field on the input to a stateless compression/decompression request can be used to pass in an initial checksum value that will be used to seed the checksums produced by the stateless operation.

The checksum algorithm CPA_DC_XXHASH32 does not support passing an input value in this parameter. Any initial value passed will be ignored by the compression/decompression operation when this checksum algorithm is used.

CpaBoolean `endOfLastBlock`

Decompression operation has stopped at the end of the last block in a deflate stream.

CpaBoolean `dataUncompressed`

If TRUE the output data for this request is uncompressed and in the format setup for the request. This value is only valid for CPA_DC_ASB_ENABLED or if compressAndVerifyAndRecover is set to TRUE in the CpaDcOpData structure for a request.

1.1.81 Struct _CpaDcSessionSetupData

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaDcSessionSetupData`

Session Setup Data.

Description: 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.

Public Members

CpaDcCompLvl `compLevel`

Compression Level from CpaDcCompLvl

CpaDcCompType `compType`

Compression type from CpaDcCompType

CpaDcHuffType `huffType`

Huffman type from CpaDcHuffType

CpaDcAutoSelectBest `autoSelectBestHuffmanTree`

Indicates if and how the implementation should select the best Huffman encoding.

CpaDcSessionDir `sessDirection`

Session direction indicating whether session is used for compression, decompression or both

CpaDcSessionState `sessState`

Session state indicating whether the session should be configured as stateless or stateful

CpaDcCompWindowSize `windowSize`

Window size from CpaDcCompWindowSize

CpaDcCompMinMatch `minMatch`

Min Match size from CpaDcCompMinMatch

CpaDcCompLZ4BlockMaxSize `lz4BlockMaxSize`

Window size from CpaDcCompLZ4BlockMaxSize

CpaBoolean `lz4BlockChecksum`

LZ4 Block Checksum setting for the LZ4 request. For LZ4 decompression operations, this setting must be set based on the B.Checksum flag originating from the LZ4 frame header. For LZ4 compression operations, this setting will be ignored as the implementation does not support generation of Data Block checksums.

CpaBoolean `lz4BlockIndependence`

LZ4 Block Independence Flag setting. For LZ4 compression operations, this setting must be set based on the Block Independence Flag originating from the LZ4 frame header. For LZ4 decompression operations, this setting is ignored. For data compressed with lz4BlockIndependence set to CPA_FALSE, it is not possible to perform parallel decompression on the compressed blocks. It is also not possible to access the produced LZ4 blocks randomly.

CpaDcChecksum `checksum`

Desired checksum required for the session

CpaBoolean `accumulateXXHash`

If TRUE the xxHash calculation for LZ4 requests using the session based APIs will be accumulated across requests, with a valid xxHash being written to *CpaDcRqResults.checksum* for the request which specifies CPA_DC_FLUSH_FINAL in *CpaDcOpData.flushFlag*. When the CPA_DC_FLUSH_FINAL is received, the internal XXHash state will be reset for this session. In the compression direction one exception is if a CPA_DC_OVERFLOW error is returned, the xxHash value in the checksum field will be valid for requests up to that point and the internal XXHash state will not be reset. This will allow a user to either create an LZ4 frame based off the data at the time of overflow, or correct the overflow condition and continue submitting requests until specifying CPA_DC_FLUSH_FINAL. Additionally the user can force the internal XXHash state to reset (even on overflow) by calling cpaDcResetXXHashState on this session. For the No-Session APIs (Ns) this flag will have no effect

1.1.82 Struct _CpaDcSessionSetupData2

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaDcSessionSetupData2`

Session Setup Data with additional Crc Control Data

Description: This structure contains data relating to setting up a session and the crc control data used for data integrity computations. The client needs to complete the information in this structure in order to setup a session.

Public Members

CpaDcSessionSetupData `dcSetupData`

Compression session setup data

CpaCrcControlData `dcSessionCrcControlData`

The Crc control data used for this session's data integrity computations

1.1.83 Struct _CpaDcSessionUpdateData

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaDcSessionUpdateData`

Session Update Data.

Description: This structure contains data relating to updating up a session. The client needs to complete the information in this structure in order to update a session.

Public Members

CpaDcCompLvl `compLevel`

Compression Level from CpaDcCompLvl

CpaDcHuffType `huffType`

Huffman type from CpaDcHuffType

CpaBoolean `enableDmm`

Desired DMM required for the session

1.1.84 Struct _CpaDcSkipData

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaDcSkipData`

Skip Region Data.

Description: This structure contains data relating to configuring skip region behaviour. A skip region is a region of an input buffer that should be omitted from processing or a region that should be inserted into the output buffer.

Public Members

CpaDcSkipMode `skipMode`

Skip mode from CpaDcSkipMode for buffer processing

Cpa32U `skipLength`

Number of bytes to skip when skip mode is enabled

Cpa32U `strideLength`

Size of the stride between skip regions when skip mode is set to CPA_DC_SKIP_STRIDE.

Cpa32U `firstSkipOffset`

Number of bytes to skip in a buffer before reading/writing the input/output data.

1.1.85 Struct _CpaDcStats

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaDcStats`

Compression Statistics Data.

Description: This structure contains data elements corresponding to statistics. Statistics are collected on a per instance basis and include: jobs submitted and completed for both compression and decompression.

Public Members

Cpa64U `numCompRequests`

Number of successful compression requests

Cpa64U `numCompRequestsErrors`

Number of compression requests that had errors and could not be processed

Cpa64U `numCompCompleted`

Compression requests completed

Cpa64U `numCompCompletedErrors`
Compression requests not completed due to errors

Cpa64U `numCompCnvErrorsRecovered`
Compression CNV errors that have been recovered

Cpa64U `numDecompRequests`
Number of successful decompression requests

Cpa64U `numDecompRequestsErrors`
Number of decompression requests that had errors and could not be processed

Cpa64U `numDecompCompleted`
Decompression requests completed

Cpa64U `numDecompCompletedErrors`
Decompression requests not completed due to errors

1.1.86 Struct _CpaFlatBuffer

- Defined in file_api_cpa.h

Struct Documentation

struct `_CpaFlatBuffer`
Flat buffer structure containing a pointer and length member.

Description: 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*.

Public Members

Cpa32U `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 *`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.

1.1.87 Struct _CpaInstanceInfo

- Defined in file_api_cpa.h

Struct Documentation

struct `_CpaInstanceInfo`

Instance Info Structure

Deprecated:

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

Description: Structure that contains the information to describe the instance.

Public Members

enum *_CpaInstanceType* `type`

Type definition for this instance.

enum *_CpaInstanceState* `state`

Operational state of the instance.

Cpa8U `name`[CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES]

Simple text string identifier for the instance.

Cpa8U `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.

1.1.88 Struct _CpaInstanceInfo2

- Defined in file_api_cpa.h

Struct Documentation

struct `_CpaInstanceInfo2`

Instance Info Structure, version 2

Description: Structure that contains the information to describe the instance.

Public Functions

`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*.

Public Members

CpaAccelerationServiceType `accelerationServiceType`

Type of service provided by this instance.

Cpa8U `vendorName`[CPA_INST_VENDOR_NAME_SIZE]

String identifying the vendor of the accelerator.

Cpa8U `partName`[CPA_INST_PART_NAME_SIZE]

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

Cpa8U `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 `instName`[CPA_INST_NAME_SIZE]

String identifying the name of the instance.

Cpa8U `instID`[CPA_INST_ID_SIZE]

String containing a unique identifier for the instance

CpaPhysicalInstanceId `physInstId`

Identifies the "physical instance" of the accelerator.

Cpa32U `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 `operState`

Operational state of the instance.

CpaBoolean `requiresPhysicallyContiguousMemory`

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

CpaBoolean `isPolled`

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

CpaBoolean `isOffloaded`

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

1.1.89 Struct _CpaIntegrityCrc

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaIntegrityCrc`

Integrity CRC calculation details

Description: This structure contains information about resulting integrity CRC calculations performed for a single request.

Public Members

Cpa32U `iCrc`

CRC calculated on request's input buffer

Cpa32U `oCrc`

CRC calculated on request's output buffer

1.1.90 Struct _CpaIntegrityCrc64b

- Defined in file_api_dc_cpa_dc.h

Struct Documentation

struct `_CpaIntegrityCrc64b`

Integrity CRC64 calculation details

Description: This structure contains information about resulting integrity CRC64 calculations performed for a single request.

Public Members

Cpa64U `iCrc`

CRC calculated on request's input buffer

Cpa64U `oCrc`

CRC calculated on request's output buffer

1.1.91 Struct _CpaPhysBufferList

- Defined in file_api_cpa.h

Struct Documentation

struct `_CpaPhysBufferList`

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

Description: 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.

Public Members

Cpa64U `reserved0`

Reserved for internal usage

Cpa32U `numBuffers`

Number of buffers in the list

Cpa32U `reserved1`

Reserved for alignment

CpaPhysFlatBuffer `flatBuffers`[]

Array of flat buffer structures, of size numBuffers

1.1.92 Struct _CpaPhysFlatBuffer

- Defined in file_api_cpa.h

Struct Documentation

struct `_CpaPhysFlatBuffer`

Flat buffer structure with physical address.

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

Public Members

Cpa32U `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 `reserved`

Reserved for alignment

CpaPhysicalAddr `bufferPhysAddr`

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

1.1.93 Struct _CpaPhysicalInstanceId

- Defined in file_api_cpa.h

Struct Documentation

struct `_CpaPhysicalInstanceId`

Physical Instance ID

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.

Description: Identifies the physical instance of an accelerator execution engine.

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.

Public Members

Cpa16U `packageId`

Identifies the package within which the accelerator is contained.

Cpa16U `acceleratorId`

Identifies the specific accelerator within the package.

Cpa16U `executionEngineId`

Identifies the specific execution engine within the accelerator.

Cpa16U `busAddress`

Identifies the bus address associated with the accelerator execution engine.

Cpa32U `kptAcHandle`

Identifies the achandle of the accelerator.

1.1.94 Struct CpaCyKptEcdsaSignRSOpData_t

- Defined in file_api_lac_cpa_cy_kpt.h

Struct Documentation

struct `CpaCyKptEcdsaSignRSOpData_t`

KPT ECDSA Sign R & S Operation Data.

File: cpa_cy_kpt.h

Key = SWK AAD = DER(OID) IV = nonce Input = (d) Encrypt (SWK, AAD, IV, (d)) Output (AuthTag, EncryptedECKey)

Description: 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. This key structure is encrypted when passed into cpaCyKptEcdsaSignRS Encrypt - AES-256-GCM (Key, AAD, Input) "||" - denotes concatenation

privatekey == EncryptedECKey || AuthTag

OID's that shall be supported by KPT implementation: Curve OID DER(OID) secp256r1 1.2.840.10045.3.1.7 06 08 2A 86 48 CE 3D 03 01 07 secp384r1 1.3.132.0.34 06 05 2B 81 04 00 22 secp521r1 1.3.132.0.35 06 05 2B 81 04 00 23

Expected private key (d) sizes: secp256r1 256 bits secp384r1 384 bits secp521r1 576 bits (rounded up to a multiple of 64-bit quadword)

AuthTag is 128 bits (16 bytes)

For optimal performance all data buffers SHOULD be 8-byte aligned.

See also:

cpaCyEcdsaSignRS()

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.

Public Members

CpaFlatBuffer `privateKey`

Encrypted private key data of the form EncryptECKey || AuthTag

CpaFlatBuffer `m`

digest of the message to be signed

1.1.95 Struct CpaCyKptLoadKey_t

- Defined in file_api_lac_cpa_cy_kpt.h

Struct Documentation

struct `CpaCyKptLoadKey_t`

KPT Loading key format specification.

Description: This structure defines the format of the symmetric wrapping key to be loaded into KPT. Application sets these parameters through the cpaCyKptLoadKey calls.

Public Members

CpaFlatBuffer `eSWK`

Encrypted SWK

CpaCyKptWrappingKeyType `wrappingAlgorithm`

Symmetric wrapping algorithm

1.1.96 Struct CpaCyKptRsaDecryptOpData_t

- Defined in file_api_lac_cpa_cy_kpt.h

Struct Documentation

struct `CpaCyKptRsaDecryptOpData_t`

KPT RSA Decryption Primitive Operation Data

File: cpa_cy_kpt.h

Description: This structure lists the different items that are required in the cpaCyKptRsaDecrypt 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 CpaCyGenFlatBufCbFunc callback function.

Note: If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyKptRsaDecrypt 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.

Public Members

CpaCyKptRsaPrivateKey *`pRecipientPrivateKey`

Pointer to the recipient's RSA private key.

CpaFlatBuffer `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.

1.1.97 Struct CpaCyKptRsaPrivateKey_t

- Defined in file_api_lac_cpa_cy_kpt.h

Struct Documentation

struct `CpaCyKptRsaPrivateKey_t`

RSA Private Key Structure.

File: cpa_cy_kpt.h

Description: 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.

Public Members

CpaCyRsaVersion `version`

Indicates the version of the PKCS #1 specification that is supported. Note that this applies to both representations.

CpaCyRsaPrivateKeyRepType `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.

CpaCyKptRsaPrivateKeyRep1 `privateKeyRep1`

This is the first representation of the RSA private key as defined in the PKCS #1 V2.2 specification.

CpaCyKptRsaPrivateKeyRep2 `privateKeyRep2`

This is the second representation of the RSA private key as defined in the PKCS #1 V2.2 specification.

1.1.98 Struct CpaCyKptRsaPrivateKeyRep1_t

- Defined in file_api_lac_cpa_cy_kpt.h

Struct Documentation

struct `CpaCyKptRsaPrivateKeyRep1_t`

RSA Private Key Structure For Representation 1.

privateKey = (EncryptedRSAKey || AuthTag)

File: cpa_cy_kpt.h

Description: 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). The representation is encrypted as follows: Encrypt - AES-256-GCM (Key, AAD, Input) "||" - denotes concatenation Key = SWK AAD = DER(OID) IV = nonce Input = (D || N) Encrypt (SWK, AAD, IV, (D || N)) Output (AuthTag, (D || N)') EncryptedRSAKey = (D || N)'

OID's that shall be supported by KPT implementation: OID DER(OID) 1.2.840.113549.1.1 06 08 2A 86 48 86 F7 0D 01 01

Permitted lengths for N and D are:

- 512 bits (64 bytes),
- 1024 bits (128 bytes),
- 1536 bits (192 bytes),
- 2048 bits (256 bytes),
- 3072 bits (384 bytes),
- 4096 bits (512 bytes), or
- 8192 bits (1024 bytes).

AuthTag is 128 bits (16 bytes)

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. It is critical a unique nonce is used for each SWK encrypt operation.

Public Members

CpaFlatBuffer `privateKey`

The EncryptedRSAKey concatenated with AuthTag

1.1.99 Struct CpaCyKptRsaPrivateKeyRep2_t

- Defined in file_api_lac_cpa_cy_kpt.h

Struct Documentation

struct `CpaCyKptRsaPrivateKeyRep2_t`

KPT RSA Private Key Structure For Representation 2.

privateKey = EncryptedRSAKey || AuthTag

File: cpa_cy_kpt.h

Description: 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. For KPT the parameters are Encrypted with the assoicated SWK as follows: Encrypt - AES-256-GCM (Key, AAD, Input) "||" - denotes concatenation Key = SWK IV = nonce AAD = DER(OID) Input = (P || Q || dP || dQ || Qinv || publicExponentE) Expanded Description: Encrypt (SWK, AAD, IV, (P || Q || dP || dQ || Qinv || publicExponentE)) EncryptedRSAKey = (P || Q || dP || dQ || Qinv || publicExponentE)' Output (AuthTag, EncryptedRSAKey)

OID's that shall be supported by KPT implementation: OID DER(OID) 1.2.840.113549.1.1 06 08 2A 86 48 86 F7 0D 01 01

All of the encrypted parameters will be of equal size. The length of each will be equal to keySize in bytes/2. For example for a key size of 256 Bytes (2048 bits), the length of P, Q, dP, dQ, and Qinv are all 128 Bytes, plus the publicExponentE of 256 Bytes, giving a total size for EncryptedRSAKey of 896 Bytes.

AuthTag is 128 bits (16 bytes)

Permitted Key Sizes are:

- 512 bits (64 bytes),
- 1024 bits (128 bytes),
- 1536 bits (192 bytes),
- 2048 bits (256 bytes),
- 3072 bits (384 bytes),
- 4096 bits (512 bytes), or
- 8192 bits (1024 bytes).

Note: It is critical a unique nonce is used for each SWK encrypt operation.

Public Members

CpaFlatBuffer `privateKey`

RSA private key representation 2 is built up from the tuple of p, q, dP, dQ, qInv, publicExponentE and AuthTag.

1.1.100 Struct CpaCyKptUnwrapContext_t

- Defined in file_api_lac_cpa_cy_kpt.h

Struct Documentation

struct `CpaCyKptUnwrapContext_t`

Structure of KPT unwrapping context.

File: cpa_cy_kpt.h

Description: This structure is a parameter of KPT crypto APIs, it contains data relating to KPT WPK unwrapping, the application needs to fill in this information.

Public Members

CpaCyKptHandle `kptHandle`

This is application's unique handle that identifies its (symmetric) wrapping key

Cpa8U `iv`[CPA_CY_KPT_MAX_IV_LENGTH]

Initialization Vector - which must be a nonce

Cpa8U `additionalAuthData`[CPA_CY_KPT_MAX_AAD_LENGTH]

A buffer holding the Additional Authenticated Data.

Cpa32U `aadLenInBytes`

Number of bytes representing the size of AAD within additionalAuthData buffer.

1.1.101 Struct CpaCyKptValidationKey_t

- Defined in file_api_lac_cpa_cy_kpt.h

Struct Documentation

struct `CpaCyKptValidationKey_t`

KPT device credentials key certificate

See also:

cpaCyKptQueryDeviceCredentials

Description: This structure defines the key format for use with KPT.

Public Members

CpaCyRsaPublicKey `publicKey`

Key

Cpa8U `signature`[CPA_CY_RSA3K_SIG_SIZE_INBYTES]

Signature of key

1.2 Enums

1.2.1 Enum _CpaAccelerationServiceType

- Defined in file_api_cpa.h

Enum Documentation

enum `_CpaAccelerationServiceType`

Service Type

Description: Enumeration of the different service types.

Values:

enumerator `CPA_ACC_SVC_TYPE_CRYPTO`

Cryptography

enumerator `CPA_ACC_SVC_TYPE_DATA_COMPRESSION`

Data Compression

enumerator `CPA_ACC_SVC_TYPE_PATTERN_MATCH`

Pattern Match

enumerator `CPA_ACC_SVC_TYPE_RAID`

RAID

enumerator `CPA_ACC_SVC_TYPE_XML`

XML

enumerator `CPA_ACC_SVC_TYPE_VIDEO_ANALYTICS`

Video Analytics

enumerator `CPA_ACC_SVC_TYPE_CRYPTO_ASYM`

Cryptography - Asymmetric service

enumerator `CPA_ACC_SVC_TYPE_CRYPTO_SYM`

Cryptography - Symmetric service

1.2.2 Enum _CpaBoolean

- Defined in file_api_cpa_types.h

Enum Documentation

enum `_CpaBoolean`

Boolean type.

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

Values:

enumerator `CPA_FALSE`

False value

enumerator `CPA_TRUE`

True value

1.2.3 Enum _CpaCyEcCurveType

- Defined in file_api_lac_cpa_cy_ec.h

Enum Documentation

enum `_CpaCyEcCurveType`

Enumeration listing curve types to use with generic multiplication and verification routines.

See also:

cpaCyEcGenericPointMultiply() cpaCyEcGenericPointVerify()

Description: This structure contains a list of different elliptic curve types. EC Point multiplication and other operations depend on the type of the curve.

Values:

enumerator `CPA_CY_EC_CURVE_TYPE_WEIERSTRASS_PRIME`

A Weierstrass curve with arithmetic in terms of the arithmetic of integers modulo p over a prime field.

enumerator `CPA_CY_EC_CURVE_TYPE_WEIERSTRASS_BINARY`

A Weierstrass curve with arithmetic in terms of operations on bits over a binary field.

enumerator `CPA_CY_EC_CURVE_TYPE_WEIERSTRASS_KOBLITZ_BINARY`

A Weierstrass-koblitz curve with arithmetic in terms of operations on the bits over a binary field.

1.2.4 Enum _CpaCyEcFieldType

- Defined in file_api_lac_cpa_cy_ec.h

Enum Documentation

enum `_CpaCyEcFieldType`

Field types for Elliptic Curve

Description: 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.

Values:

enumerator `CPA_CY_EC_FIELD_TYPE_PRIME`

A prime field, GF(p)

enumerator `CPA_CY_EC_FIELD_TYPE_BINARY`

A binary field, GF(2^m)

1.2.5 Enum _CpaCyEcMontEdwdsCurveType

- Defined in file_api_lac_cpa_cy_ec.h

Enum Documentation

enum `_CpaCyEcMontEdwdsCurveType`

Curve types for Elliptic Curves defined in RFC#7748

Description: As defined by RFC 7748, there are four elliptic curves in this group. The Montgomery curves are denoted curve25519 and curve448, and the birationally equivalent Twisted Edwards curves are denoted edwards25519 and edwards448

Values:

enumerator `CPA_CY_EC_MONTEDWDS_CURVE25519_TYPE`

Montgomery 25519 curve

enumerator `CPA_CY_EC_MONTEDWDS_ED25519_TYPE`

Edwards 25519 curve

enumerator `CPA_CY_EC_MONTEDWDS_CURVE448_TYPE`

Montgomery 448 curve

enumerator `CPA_CY_EC_MONTEDWDS_ED448_TYPE`

Edwards 448 curve

1.2.6 Enum _CpaCyKeyHKDFCipherSuite

- Defined in file_api_lac_cpa_cy_key.h

Enum Documentation

enum `_CpaCyKeyHKDFCipherSuite`

TLS Operation Types

The function *cpaCyKeyGenTls3* accelerates the TLS HKDF, which is defined as part of RFC5869 (HKDF) and RFC8446 (TLS v1.3).

File: cpa_cy_key.h

Description: Enumeration of the different cipher suites that may be used in a TLS v1.3 operation. This value is used to infer the sizes of the key and iv sublabel.

This enumerated type defines the supported cipher suites in the TLS operation that require HKDF key operations.

Values:

enumerator `CPA_CY_HKDF_TLS_AES_128_GCM_SHA256`

enumerator `CPA_CY_HKDF_TLS_AES_256_GCM_SHA384`

enumerator `CPA_CY_HKDF_TLS_CHACHA20_POLY1305_SHA256`

enumerator `CPA_CY_HKDF_TLS_AES_128_CCM_SHA256`

enumerator `CPA_CY_HKDF_TLS_AES_128_CCM_8_SHA256`

1.2.7 Enum _CpaCyKeyHKDFOp

- Defined in file_api_lac_cpa_cy_key.h

Enum Documentation

enum `_CpaCyKeyHKDFOp`

TLS Operation Types

The function *cpaCyKeyGenTls3* accelerates the TLS HKDF, which is defined as part of RFC5869 (HKDF) and RFC8446 (TLS v1.3).

File: cpa_cy_key.h

Description: Enumeration of the different TLS operations that can be specified in the CpaCyKey-GenHKDFOpData.

This enumerated type defines the support HKDF operations for extraction and expansion of keying material.

Values:

enumerator `CPA_CY_HKDF_KEY_EXTRACT`

HKDF Extract operation Corresponds to RFC5869 section 2.2, step 1 "Extract"

enumerator `CPA_CY_HKDF_KEY_EXPAND`

HKDF Expand operation Corresponds to RFC5869 section 2.3, step 2 "Expand"

enumerator `CPA_CY_HKDF_KEY_EXTRACT_EXPAND`

HKDF operation This performs HKDF_EXTRACT and HKDF_EXPAND in a single API invocation.

enumerator `CPA_CY_HKDF_KEY_EXPAND_LABEL`

HKDF Expand label operation for TLS 1.3 Corresponds to RFC8446 section 7.1 Key Schedule definition for HKDF-Expand-Label, which refers to HKDF-Expand defined in RFC5869.

enumerator `CPA_CY_HKDF_KEY_EXTRACT_EXPAND_LABEL`

HKDF Extract plus Expand label operation for TLS 1.3 Corresponds to RFC5869 section 2.2, step 1 "Extract" followed by RFC8446 section 7.1 Key Schedule definition for HKDF-Expand-Label, which refers to HKDF-Expand defined in RFC5869.

1.2.8 Enum _CpaCyKeySslOp

- Defined in file_api_lac_cpa_cy_key.h

Enum Documentation

enum `_CpaCyKeySslOp`

SSL Operation Types

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

Values:

enumerator `CPA_CY_KEY_SSL_OP_MASTER_SECRET_DERIVE`

Derive the master secret

enumerator `CPA_CY_KEY_SSL_OP_KEY_MATERIAL_DERIVE`

Derive the key material

enumerator `CPA_CY_KEY_SSL_OP_USER_DEFINED`

User Defined Operation for custom labels

1.2.9 Enum _CpaCyKeyTlsOp

- Defined in file_api_lac_cpa_cy_key.h

Enum Documentation

enum `_CpaCyKeyTlsOp`

TLS Operation Types

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 each of 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.

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

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.

Values:

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".

enumerator `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".

enumerator `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".

enumerator `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".

enumerator `CPA_CY_KEY_TLS_OP_USER_DEFINED`

User Defined Operation for custom labels.

1.2.10 Enum _CpaCyPriority

- Defined in file_api_lac_cpa_cy_common.h

Enum Documentation

enum `_CpaCyPriority`

Request priority

File: cpa_cy_common.h

Description: 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.

Values:

enumerator `CPA_CY_PRIORITY_NORMAL`

Normal priority

enumerator `CPA_CY_PRIORITY_HIGH`

High priority

1.2.11 Enum _CpaCyRsaPrivateKeyRepType

- Defined in file_api_lac_cpa_cy_rsa.h

Enum Documentation

enum `_CpaCyRsaPrivateKeyRepType`

RSA private key representation type.

Description: This enumeration lists which PKCS V2.1 representation of the private key is being used.

Values:

enumerator `CPA_CY_RSA_PRIVATE_KEY_REP_TYPE_1`

The first representation of the RSA private key.

enumerator `CPA_CY_RSA_PRIVATE_KEY_REP_TYPE_2`

The second representation of the RSA private key.

1.2.12 Enum _CpaCyRsaVersion

- Defined in file_api_lac_cpa_cy_rsa.h

Enum Documentation

enum `_CpaCyRsaVersion`

RSA Version.

Description: This enumeration lists the version identifier for the PKCS #1 V2.1 standard.

Note: Multi-prime (more than two primes) is not supported.

Values:

enumerator `CPA_CY_RSA_VERSION_TWO_PRIME`

The version supported is "two-prime".

1.2.13 Enum _CpaCySymAlgChainOrder

- Defined in file_api_lac_cpa_cy_sym.h

Enum Documentation

enum `_CpaCySymAlgChainOrder`

Algorithm Chaining Operation Ordering

Description: This enum defines the ordering of operations for algorithm chaining.

Values:

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                |
+------------------------------------------+
```

enumerator 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 |
+--------+---------------------------+--------+------+
```

1.2.14 Enum _CpaCySymCipherAlgorithm

- Defined in file_api_lac_cpa_cy_sym.h

Enum Documentation

enum `_CpaCySymCipherAlgorithm`

Cipher algorithms.

Description: This enumeration lists supported cipher algorithms and modes.

Values:

enumerator **`CPA_CY_SYM_CIPHER_NULL`**

NULL cipher algorithm. No mode applies to the NULL algorithm.

enumerator **`CPA_CY_SYM_CIPHER_ARC4`**

(A)RC4 cipher algorithm

enumerator **`CPA_CY_SYM_CIPHER_AES_ECB`**

AES algorithm in ECB mode

enumerator **`CPA_CY_SYM_CIPHER_AES_CBC`**

AES algorithm in CBC mode

enumerator **`CPA_CY_SYM_CIPHER_AES_CTR`**

AES algorithm in Counter mode

enumerator **`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.

enumerator **`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.

enumerator `CPA_CY_SYM_CIPHER_DES_ECB`

DES algorithm in ECB mode

enumerator `CPA_CY_SYM_CIPHER_DES_CBC`

DES algorithm in CBC mode

enumerator `CPA_CY_SYM_CIPHER_3DES_ECB`

Triple DES algorithm in ECB mode

enumerator `CPA_CY_SYM_CIPHER_3DES_CBC`

Triple DES algorithm in CBC mode

enumerator `CPA_CY_SYM_CIPHER_3DES_CTR`

Triple DES algorithm in CTR mode

enumerator `CPA_CY_SYM_CIPHER_KASUMI_F8`

Kasumi algorithm in F8 mode

enumerator `CPA_CY_SYM_CIPHER_SNOW3G_UEA2`

SNOW3G algorithm in UEA2 mode

enumerator `CPA_CY_SYM_CIPHER_AES_F8`

AES algorithm in F8 mode

enumerator `CPA_CY_SYM_CIPHER_AES_XTS`

AES algorithm in XTS mode

enumerator `CPA_CY_SYM_CIPHER_ZUC_EEA3`

ZUC algorithm in EEA3 mode

enumerator `CPA_CY_SYM_CIPHER_CHACHA`

ChaCha20 Cipher Algorithm. This cipher is only supported for algorithm chaining. When selected, the hash algorithm must be set to CPA_CY_SYM_HASH_POLY and the hash mode must be set to CPA_CY_SYM_HASH_MODE_AUTH.

enumerator `CPA_CY_SYM_CIPHER_SM4_ECB`

SM4 algorithm in ECB mode This cipher supports 128 bit keys only and does not support partial processing.

enumerator `CPA_CY_SYM_CIPHER_SM4_CBC`

SM4 algorithm in CBC mode This cipher supports 128 bit keys only and does not support partial processing.

enumerator `CPA_CY_SYM_CIPHER_SM4_CTR`

SM4 algorithm in CTR mode This cipher supports 128 bit keys only and does not support partial processing.

1.2.15 Enum _CpaCySymCipherDirection

- Defined in file_api_lac_cpa_cy_sym.h

Enum Documentation

enum `_CpaCySymCipherDirection`

Symmetric Cipher Direction

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

Values:

enumerator `CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT`

Encrypt Data

enumerator `CPA_CY_SYM_CIPHER_DIRECTION_DECRYPT`

Decrypt Data

1.2.16 Enum _CpaCySymHashAlgorithm

- Defined in file_api_lac_cpa_cy_sym.h

Enum Documentation

enum `_CpaCySymHashAlgorithm`

Hash algorithms.

Description: This enumeration lists supported hash algorithms.

Values:

enumerator `CPA_CY_SYM_HASH_NONE`

No hash algorithm.

enumerator `CPA_CY_SYM_HASH_MD5`

MD5 algorithm. Supported in all 3 hash modes

enumerator `CPA_CY_SYM_HASH_SHA1`

128 bit SHA algorithm. Supported in all 3 hash modes

enumerator `CPA_CY_SYM_HASH_SHA224`

224 bit SHA algorithm. Supported in all 3 hash modes

enumerator `CPA_CY_SYM_HASH_SHA256`

256 bit SHA algorithm. Supported in all 3 hash modes

enumerator `CPA_CY_SYM_HASH_SHA384`

384 bit SHA algorithm. Supported in all 3 hash modes

enumerator `CPA_CY_SYM_HASH_SHA512`

512 bit SHA algorithm. Supported in all 3 hash modes

enumerator `CPA_CY_SYM_HASH_AES_XCBC`

AES XCBC algorithm. This is only supported in the hash mode CPA_CY_SYM_HASH_MODE_AUTH.

enumerator `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.

enumerator `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.

enumerator `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.

enumerator **`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.

enumerator **`CPA_CY_SYM_HASH_AES_CMAC`**

AES CMAC algorithm. This is only supported in the hash mode CPA_CY_SYM_HASH_MODE_AUTH.

enumerator **`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.

enumerator **`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.

enumerator **`CPA_CY_SYM_HASH_ZUC_EIA3`**

ZUC algorithm in EIA3 mode

enumerator **`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 supported for this algorithm. Partial requests are not supported, that is, only requests of CPA_CY_SYM_PACKET_TYPE_FULL are supported.

enumerator **`CPA_CY_SYM_HASH_SHA3_224`**

224 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 supported for this algorithm.

enumerator **`CPA_CY_SYM_HASH_SHA3_384`**

384 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 supported for this algorithm. Partial requests are not supported, that is, only requests of CPA_CY_SYM_PACKET_TYPE_FULL are supported.

enumerator **`CPA_CY_SYM_HASH_SHA3_512`**

512 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 supported for this algorithm. Partial requests are not supported, that is, only requests of CPA_CY_SYM_PACKET_TYPE_FULL are supported.

enumerator **`CPA_CY_SYM_HASH_SHAKE_128`**

128 bit SHAKE algorithm. This is only supported in the hash mode CPA_CY_SYM_HASH_MODE_PLAIN. Partial requests are not supported, that is, only requests of CPA_CY_SYM_PACKET_TYPE_FULL are supported.

enumerator **`CPA_CY_SYM_HASH_SHAKE_256`**

256 bit SHAKE algorithm. This is only supported in the hash mode CPA_CY_SYM_HASH_MODE_PLAIN. Partial requests are not supported, that is, only requests of CPA_CY_SYM_PACKET_TYPE_FULL are supported.

enumerator **`CPA_CY_SYM_HASH_POLY`**

Poly1305 hash algorithm. This is only supported in the hash mode CPA_CY_SYM_HASH_MODE_AUTH. This hash algorithm is only supported as part of an algorithm chain with AES_CY_SYM_CIPHER_CHACHA to implement the ChaCha20-Poly1305 AEAD algorithm.

enumerator **`CPA_CY_SYM_HASH_SM3`**

SM3 hash algorithm. Supported in all 3 hash modes.

1.2.17 Enum _CpaCySymHashMode

- Defined in file_api_lac_cpa_cy_sym.h

Enum Documentation

enum **`_CpaCySymHashMode`**

Symmetric Hash mode

Description: This enum indicates the Hash Mode.

Values:

enumerator **`CPA_CY_SYM_HASH_MODE_PLAIN`**

Plain hash. Can be specified for MD5 and the SHA family of hash algorithms.

enumerator **`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.

enumerator `CPA_CY_SYM_HASH_MODE_NESTED`

Nested hash. Can be specified for MD5 and the SHA family of hash algorithms.

1.2.18 Enum _CpaCySymOp

- Defined in file_api_lac_cpa_cy_sym.h

Enum Documentation

enum `_CpaCySymOp`

Types of operations supported by the cpaCySymPerformOp function.

See also:

cpaCySymPerformOp

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

Values:

enumerator `CPA_CY_SYM_OP_NONE`

No operation

enumerator `CPA_CY_SYM_OP_CIPHER`

Cipher only operation on the data

enumerator `CPA_CY_SYM_OP_HASH`

Hash only operation on the data

enumerator `CPA_CY_SYM_OP_ALGORITHM_CHAINING`

Chain any cipher with any hash operation. The order depends on the value in the CpaCySy-mAlgChainOrder 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*.

1.2.19 Enum _CpaCySymPacketType

- Defined in file_api_lac_cpa_cy_sym.h

Enum Documentation

enum `_CpaCySymPacketType`

Packet type for the cpaCySymPerformOp function

See also:

cpaCySymPerformOp()

Description: 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*.

Values:

enumerator `CPA_CY_SYM_PACKET_TYPE_FULL`

Perform an operation on a full packet

enumerator `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.

enumerator `CPA_CY_SYM_PACKET_TYPE_LAST_PARTIAL`

Complete the last part of a multi-part operation

1.2.20 Enum _CpaDcAutoSelectBest

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcAutoSelectBest`

Supported modes for automatically selecting the best compression type.

When CPA_DC_ASB_ENABLED is used the output will be a format compliant block, whether the data is compressed or not.

Description: This enumeration lists the supported modes for automatically selecting the best encoding which would lead to the best compression results.

The following values are deprecated and should not be used. They will be removed in a future version of this file.

- CPA_DC_ASB_STATIC_DYNAMIC
- CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_STORED_HDRS
- CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_NO_HDRS

Values:

enumerator `CPA_DC_ASB_DISABLED`

Auto select best mode is disabled

enumerator `CPA_DC_ASB_STATIC_DYNAMIC`

Auto select between static and dynamic compression

enumerator `CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_STORED_HDRS`

Auto select between uncompressed, static and dynamic compression, using stored block deflate headers if uncompressed is selected

enumerator `CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_NO_HDRS`

Auto select between uncompressed, static and dynamic compression, using no deflate headers if uncompressed is selected

enumerator `CPA_DC_ASB_ENABLED`

Auto select best mode is enabled

1.2.21 Enum _CpaDcChainOperations

- Defined in file_api_dc_cpa_dc_chain.h

Enum Documentation

enum `_CpaDcChainOperations`

Supported operations for compression chaining

Description: This enumeration lists the supported operations for compression chaining

Values:

enumerator `CPA_DC_CHAIN_COMPRESS_THEN_HASH`

2 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform hash on compressed text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for hash setup data

enumerator `CPA_DC_CHAIN_COMPRESS_THEN_ENCRYPT`

2 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform encryption on compressed text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for encryption setup data

enumerator `CPA_DC_CHAIN_COMPRESS_THEN_HASH_ENCRYPT`

2 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform hash on compressed text and encryption on compressed text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for hash and encryption setup data

enumerator `CPA_DC_CHAIN_COMPRESS_THEN_ENCRYPT_HASH`

2 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform encryption on compressed text and hash on compressed & encrypted text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for encryption and hash setup data

enumerator `CPA_DC_CHAIN_COMPRESS_THEN_AEAD`

2 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform AEAD encryption on compressed text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for AEAD encryption setup data

enumerator `CPA_DC_CHAIN_HASH_THEN_COMPRESS`

2 operations for chaining: 1st operation is to perform hash on plain text 2nd operation is to perform compression on plain text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for hash setup data 2nd entry is for compression setup data

enumerator `CPA_DC_CHAIN_HASH_VERIFY_THEN_DECOMPRESS`

2 operations for chaining: 1st operation is to perform hash verify on compressed text 2nd operation is to perform decompression on compressed text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for hash setup data 2nd entry is for decompression setup data

enumerator `CPA_DC_CHAIN_DECRYPT_THEN_DECOMPRESS`

2 operations for chaining: 1st operation is to perform decryption on compressed & encrypted text 2nd operation is to perform decompression on compressed text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for decryption setup data 2nd entry is for decompression setup data

enumerator `CPA_DC_CHAIN_HASH_VERIFY_DECRYPT_THEN_DECOMPRESS`

2 operations for chaining: 1st operation is to perform hash verify on compressed & encrypted text and decryption on compressed & encrypted text 2nd operation is to perform decompression on compressed text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for hash and decryption setup data 2nd entry is for decompression setup data

enumerator `CPA_DC_CHAIN_DECRYPT_HASH_VERIFY_THEN_DECOMPRESS`

2 operations for chaining: 1st operation is to perform decryption on compressed & encrypted text and hash verify on compressed text 2nd operation is to perform decompression on compressed text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for decryption and hash setup data 2nd entry is for decompression setup data

enumerator `CPA_DC_CHAIN_AEAD_THEN_DECOMPRESS`

2 operations for chaining: 1st operation is to perform AEAD decryption on compressed & encrypted text 2nd operation is to perform decompression on compressed text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for AEAD decryption setup data 2nd entry is for decompression setup data

enumerator `CPA_DC_CHAIN_DECOMPRESS_THEN_HASH_VERIFY`

2 operations for chaining: 1st operation is to perform decompression on compressed text 2nd operation is to perform hash verify on plain text < 2 entries in CpaDcChainSessionSetupData array: 1st entry is for decompression setup data 2nd entry is for hash setup data

enumerator `CPA_DC_CHAIN_COMPRESS_THEN_AEAD_THEN_HASH`

3 operations for chaining: 1st operation is to perform compression on plain text 2nd operation is to perform AEAD encryption compressed text 3rd operation is to perform hash on compressed & encrypted text < 3 entries in CpaDcChainSessionSetupData array: 1st entry is for compression setup data 2nd entry is for AEAD encryption setup data 3rd entry is for hash setup data

1.2.22 Enum _CpaDcChainSessionType

- Defined in file_api_dc_cpa_dc_chain.h

Enum Documentation

enum `_CpaDcChainSessionType`

Supported session types for data compression chaining.

Description: This enumeration lists the supported session types for data compression chaining.

Values:

enumerator `CPA_DC_CHAIN_COMPRESS_DECOMPRESS`

Indicate the session is for compression or decompression

enumerator `CPA_DC_CHAIN_SYMMETRIC_CRYPTO`

Indicate the session is for symmetric crypto

1.2.23 Enum _CpaDcChecksum

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcChecksum`

Supported checksum algorithms

Description: This enumeration lists the supported checksum algorithms Used to decide on file header and footer specifics.

Values:

enumerator `CPA_DC_NONE`

No checksum required

enumerator `CPA_DC_CRC32`

Application requires a CRC32 checksum

enumerator `CPA_DC_ADLER32`

Application requires Adler-32 checksum

enumerator `CPA_DC_CRC32_ADLER32`

Application requires both CRC32 and Adler-32 checksums

enumerator `CPA_DC_XXHASH32`

Application requires xxHash-32 checksum

1.2.24 Enum _CpaDcCompLvl

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcCompLvl`

Supported compression levels

Description: This enumerated lists the supported compressed levels. Lower values will result in less compressibility in less time.

Values:

enumerator `CPA_DC_L1`

Compression level 1

enumerator `CPA_DC_L2`

Compression level 2

enumerator `CPA_DC_L3`

Compression level 3

enumerator `CPA_DC_L4`

Compression level 4

enumerator `CPA_DC_L5`

Compression level 5

enumerator `CPA_DC_L6`

Compression level 6

enumerator `CPA_DC_L7`

Compression level 7

enumerator `CPA_DC_L8`

Compression level 8

enumerator `CPA_DC_L9`

Compression level 9

enumerator `CPA_DC_L10`

Compression level 10

enumerator `CPA_DC_L11`

Compression level 11

enumerator `CPA_DC_L12`

Compression level 12

1.2.25 Enum _CpaDcCompLZ4BlockMaxSize

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcCompLZ4BlockMaxSize`

Maximum LZ4 output block size

Description: Maximum LZ4 output block size

Values:

enumerator `CPA_DC_LZ4_MAX_BLOCK_SIZE_64K`

Maximum block size 64K

enumerator `CPA_DC_LZ4_MAX_BLOCK_SIZE_256K`

Maximum block size 256K

enumerator `CPA_DC_LZ4_MAX_BLOCK_SIZE_1M`

Maximum block size 1M

enumerator `CPA_DC_LZ4_MAX_BLOCK_SIZE_4M`

Maximum block size 4M

1.2.26 Enum _CpaDcCompMinMatch

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcCompMinMatch`

Min match size in bytes

Description: This is the min match size that will be used for the search algorithm. It is only configurable for LZ4S.

Values:

enumerator `CPA_DC_MIN_3_BYTE_MATCH`

Min Match of 3 bytes

enumerator `CPA_DC_MIN_4_BYTE_MATCH`

Min Match of 4 bytes

1.2.27 Enum _CpaDcCompType

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcCompType`

Supported compression types

Description: This enumeration lists the supported data compression algorithms. In combination with CpaDcChecksum it is used to decide on the file header and footer format.

Values:

enumerator `CPA_DC_DEFLATE`

Deflate Compression

enumerator `CPA_DC_LZ4`

LZ4 Compression

enumerator `CPA_DC_LZ4S`

LZ4S Compression

1.2.28 Enum _CpaDcCompWindowSize

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcCompWindowSize`

Support for defined algorithm window sizes

Description: This enumerated list defines the valid window sizes that can be used with the supported algorithms

Values:

enumerator `CPA_DC_WINSIZE_4K`

Window size of 4KB

enumerator `CPA_DC_WINSIZE_8K`

Window size of 8KB

enumerator `CPA_DC_WINSIZE_16K`

Window size of 16KB

enumerator `CPA_DC_WINSIZE_32K`

Window size of 32KB

1.2.29 Enum _CpaDcFlush

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcFlush`

Supported flush flags

Description: This enumerated list identifies the types of flush that can be specified for stateful and stateless cpaDcCompressData and cpaDcDecompressData functions.

Values:

enumerator `CPA_DC_FLUSH_NONE`

No flush request.

enumerator `CPA_DC_FLUSH_FINAL`

Indicates that the input buffer contains all of the data for the compression session allowing any buffered data to be released. For Deflate, BFINAL is set in the compression header.

enumerator `CPA_DC_FLUSH_SYNC`

Used for stateful deflate compression to indicate that all pending output is flushed, byte aligned, to the output buffer. The session state is not reset.

enumerator `CPA_DC_FLUSH_FULL`

Used for deflate compression to indicate that all pending output is flushed to the output buffer and the session state is reset.

1.2.30 Enum _CpaDcHuffType

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcHuffType`

Supported Huffman Tree types

Selecting Full Dynamic Huffman trees generates compressed blocks with an RFC 1951 header specifying "compressed with dynamic Huffman codes". The headers are calculated on the data being compressed, requiring two passes.

Description: This enumeration lists support for Huffman Tree types. Selecting Static Huffman trees generates compressed blocks with an RFC 1951 header specifying "compressed with fixed Huffman trees".

Selecting Precompiled Huffman Trees generates blocks with RFC 1951 dynamic headers. The headers are pre-calculated and are specified by the file type.

Values:

enumerator `CPA_DC_HT_STATIC`

Static Huffman Trees

enumerator `CPA_DC_HT_PRECOMP`

Precompiled Huffman Trees

enumerator `CPA_DC_HT_FULL_DYNAMIC`

Full Dynamic Huffman Trees

1.2.31 Enum _CpaDcIntegrityCrcSize

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcIntegrityCrcSize`

Integrity CRC Size

Description: Enum of possible integrity CRC sizes.

Values:

enumerator `CPA_DC_INTEGRITY_CRC32`

32-bit Integrity CRCs

enumerator `CPA_DC_INTEGRITY_CRC64`

64-bit integrity CRCs

1.2.32 Enum _CpaDcReqStatus

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcReqStatus`

Supported additional details from accelerator

Description: This enumeration lists the supported additional details from the accelerator. These may be useful in determining the best way to recover from a failure.

Values:

enumerator `CPA_DC_OK`

No error detected by compression slice

enumerator `CPA_DC_INVALID_BLOCK_TYPE`

Invalid block type (type == 3)

enumerator `CPA_DC_BAD_STORED_BLOCK_LEN`

Stored block length did not match one's complement

enumerator `CPA_DC_TOO_MANY_CODES`

Too many length or distance codes

enumerator `CPA_DC_INCOMPLETE_CODE_LENS`

Code length codes incomplete

enumerator `CPA_DC_REPEATED_LENS`

Repeated lengths with no first length

enumerator `CPA_DC_MORE_REPEAT`

Repeat more than specified lengths

enumerator `CPA_DC_BAD_LITLEN_CODES`

Invalid literal/length code lengths

enumerator `CPA_DC_BAD_DIST_CODES`

Invalid distance code lengths

enumerator `CPA_DC_INVALID_CODE`

Invalid literal/length or distance code in fixed or dynamic block

enumerator `CPA_DC_INVALID_DIST`

Distance is too far back in fixed or dynamic block

enumerator `CPA_DC_OVERFLOW`

Overflow detected. This is an indication that output buffer has overflowed. For stateful sessions, this is a warning (the input can be adjusted and resubmitted). For stateless sessions this is an error condition

enumerator `CPA_DC_SOFTERR`

Other non-fatal detected

enumerator `CPA_DC_FATALERR`

Fatal error detected

enumerator `CPA_DC_MAX_RESUBITERR`

On an error being detected, the firmware attempted to correct and resubmitted the request, however, the maximum resubmit value was exceeded

enumerator `CPA_DC_INCOMPLETE_FILE_ERR`

The input file is incomplete. Note this is an indication that the request was submitted with a CPA_DC_FLUSH_FINAL, however, a BFINAL bit was not found in the request

enumerator `CPA_DC_WDOG_TIMER_ERR`

The request was not completed as a watchdog timer hardware event occurred

enumerator `CPA_DC_EP_HARDWARE_ERR`

Request was not completed as an end point hardware error occurred (for example, a parity error)

enumerator `CPA_DC_VERIFY_ERROR`

Error detected during "compress and verify" operation enumerator `CPA_DC_EMPTY_DYM_BLK`

Decompression request contained an empty dynamic stored block (not supported)

enumerator `CPA_DC_CRC_INTEG_ERR`

A data integrity CRC error was detected

enumerator `CPA_DC_LZ4_MAX_BLOCK_SIZE_EXCEEDED`

LZ4 max block size exceeded

enumerator `CPA_DC_LZ4_BLOCK_OVERFLOW_ERR`

LZ4 Block Overflow Error

enumerator `CPA_DC_LZ4_TOKEN_IS_ZERO_ERR`

LZ4 Decoded token offset or token length is zero

enumerator `CPA_DC_LZ4_DISTANCE_OUT_OF_RANGE_ERR`

LZ4 Distance out of range for len/distance pair

1.2.33 Enum _CpaDcSessionDir

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcSessionDir`

Supported session directions

Description: This enumerated list identifies the direction of a session. A session can be compress, decompress or both.

Values:

enumerator `CPA_DC_DIR_COMPRESS`

Session will be used for compression

enumerator `CPA_DC_DIR_DECOMPRESS`

Session will be used for decompression

enumerator `CPA_DC_DIR_COMBINED`

Session will be used for both compression and decompression

1.2.34 Enum _CpaDcSessionState

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcSessionState`

Supported session state settings

Stateful sessions are limited to have only one in-flight message per session. This means a compress or decompress request must be complete before a new request can be started. This applies equally to sessions that are uni-directional in nature and sessions that are combined compress and decompress. Completion occurs when the synchronous function returns, or when the asynchronous callback function has completed.

Description: This enumerated list identifies the stateful setting of a session. A session can be either stateful or stateless.

Values:

enumerator `CPA_DC_STATEFUL`

Session will be stateful, implying that state may need to be saved in some situations

enumerator `CPA_DC_STATELESS`

Session will be stateless, implying no state will be stored

1.2.35 Enum _CpaDcSkipMode

- Defined in file_api_dc_cpa_dc.h

Enum Documentation

enum `_CpaDcSkipMode`

Supported modes for skipping regions of input or output buffers.

Description: This enumeration lists the supported modes for skipping regions of input or output buffers.

Values:

enumerator `CPA_DC_SKIP_DISABLED`

Skip mode is disabled

enumerator `CPA_DC_SKIP_AT_START`

Skip region is at the start of the buffer.

enumerator `CPA_DC_SKIP_AT_END`

Skip region is at the end of the buffer.

enumerator `CPA_DC_SKIP_STRIDE`

Skip region occurs at regular intervals within the buffer. *CpaDcSkipData.strideLength* specifies the number of bytes between each skip region.

1.2.36 Enum _CpaInstanceAllocPolicy

- Defined in file_api_cpa.h

Enum Documentation

enum `_CpaInstanceAllocPolicy`

Instance Allocation Policies

Description: Enumeration of the possible instance allocation policies that may be used for allocating instances using the *cpaAllocInstance()* API.

Values:

enumerator `CPA_INST_ALLOC_NONE`

No policy specified. The implementation will choose a default allocation scheme which may be device dependent.

enumerator `CPA_INST_ALLOC_PREFER_EXISTING`

Allocate new instances from the same underlying hardware devices that are already in use by the same process that is requesting the new allocation. If no instances are available, a new device, if available, will be used. Note, "already in use" means in use for any service, not specifically for the service type requested. The service type requested is only used to determine which of the available underlying devices can support the service, not to group instance allocations based on service type.

enumerator `CPA_INST_ALLOC_PREFER_NEW`

Allocate new instances from a device that is not currently in use. If no unused devices are available, existing devices used by the process requesting the allocation may be used.

1.2.37 Enum _CpaInstanceEvent

- Defined in file_api_cpa.h

Enum Documentation

enum `_CpaInstanceEvent`

Instance Events

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

Values:

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.

enumerator `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.

enumerator `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.

1.2.38 Enum _CpaInstanceState

- Defined in file_api_cpa.h

Enum Documentation

enum `_CpaInstanceState`

Instance State

Deprecated:

As of v1.3 of the Crypto API, this enum has been deprecated, replaced by *CpaOperational-State*.

Description: Enumeration of the different instance states that are possible.

Values:

enumerator `CPA_INSTANCE_STATE_INITIALISED`

Instance is in the initialized state and ready for use.

enumerator `CPA_INSTANCE_STATE_SHUTDOWN`

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

1.2.39 Enum _CpaInstanceType

- Defined in file_api_cpa.h

Enum Documentation

enum `_CpaInstanceType`

Instance Types

Deprecated:

As of v1.3 of the Crypto API, this enum has been deprecated, replaced by *CpaAccelerationSer-viceType*.

Description: Enumeration of the different instance types.

Values:

enumerator `CPA_INSTANCE_TYPE_CRYPTO`

Cryptographic instance type

enumerator `CPA_INSTANCE_TYPE_DATA_COMPRESSION`

Data compression instance type

enumerator `CPA_INSTANCE_TYPE_RAID`

RAID instance type

enumerator `CPA_INSTANCE_TYPE_XML`

XML instance type

enumerator `CPA_INSTANCE_TYPE_REGEX`

Regular Expression instance type

1.2.40 Enum _CpaOperationalState

- Defined in file_api_cpa.h

Enum Documentation

enum `_CpaOperationalState`

Instance operational state

Description: Enumeration of the different operational states that are possible.

Values:

enumerator `CPA_OPER_STATE_DOWN`

Instance is not available for use. May not yet be initialized, or stopped.

enumerator `CPA_OPER_STATE_UP`

Instance is available for use. Has been initialized and started.

1.2.41 Enum CpaCyKptKeyManagementStatus_t

- Defined in file_api_lac_cpa_cy_kpt.h

Enum Documentation

enum `CpaCyKptKeyManagementStatus_t`

Return Status

Description: This enumeration lists all the possible return status after completing KPT APIs.

Values:

enumerator `CPA_CY_KPT_SUCCESS`

Generic success status for all KPT wrapping key handling functions

enumerator `CPA_CY_KPT_LOADKEY_FAIL_QUOTA_EXCEEDED_PER_VFID`

SWK count exceeds the configured maxmium value per VFID

enumerator `CPA_CY_KPT_LOADKEY_FAIL_QUOTA_EXCEEDED_PER_PASID`

SWK count exceeds the configured maxmium value per PASID

enumerator `CPA_CY_KPT_LOADKEY_FAIL_QUOTA_EXCEEDED`

SWK count exceeds the configured maxmium value when not scoped to VFID or PASID

enumerator `CPA_CY_KPT_SWK_FAIL_NOT_FOUND`

Unable to find SWK entry by handle

enumerator `CPA_CY_KPT_FAILED`

1.2.42 Enum CpaCyKptWrappingKeyType_t

- Defined in file_api_lac_cpa_cy_kpt.h

Enum Documentation

enum `CpaCyKptWrappingKeyType_t`

Cipher algorithms used to generate a wrapped private key (WPK) from the clear private key.

Description: This enumeration lists supported cipher algorithms and modes.

Values:

enumerator `CPA_CY_KPT_WRAPPING_KEY_TYPE_AES256_GCM`

1.3 Unions

1.3.1 Union _CpaCyEcCurveParameters

- Defined in file_api_lac_cpa_cy_ec.h

Union Documentation

union `_CpaCyEcCurveParameters`

#include <cpa_cy_ec.h> Union characterised by a specific curve.

See also:

CpaCyEcCurveParametersWeierstrass

Description: This union allows for the characterisation of different curve types encapsulted in one data type. The intention is that new curve types will be added in the future.

Note:

Public Members

CpaCyEcCurveParametersWeierstrass `weierstrassParameters`

1.4 Functions

1.4.1 Function cpaAllocInstance

- Defined in file_api_cpa.h

Function Documentation

CpaStatus `cpaAllocInstance`(const *CpaAccelerationServiceType* serviceType, const *CpaInstanceAllocPolicy* policy, *CpaInstanceHandle* *pInstanceHandle)

Allocate a service instance.

When this function is called in user space, the implementation may only be able to allocate from the devices that are visible to the user space process context from which the function is called.

Description: This function is used to allocate a service instance. If there are multiple devices with free instances, the allocation policy is used to influence which device the instance will be allocated from. If multiple instances are required, this function must be called multiple times to allocate each instance.

See also:

None

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

- `serviceType` – **[in]** Type of acceleration service instance to allocate.
- `policy` – **[in]** Allocation policy

- `pInstanceHandle` – **[out]** Allocated instance or NULL if the allocation failed and no instance was allocated.

Return values

- `CPA_STATUS_SUCCESS` – An instance was successfully allocated.
- `CPA_STATUS_FAIL` – Function failed to allocate an instance.
- `CPA_STATUS_INVALID_PARAM` – Invalid parameter passed in.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.

Pre None

Post None

1.4.2 Function cpaCyBufferListGetMetaSize

- Defined in file_api_lac_cpa_cy_common.h

Function Documentation

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

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

See also:

cpaCyGetInstances()

Description: 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

Note: None

Parameters

- `instanceHandle` – **[in]** Handle to an instance of this API.
- `numBuffers` – **[in]** The number of pointers in the CpaBufferList. this is the maximum number of CpaFlatBuffers which may be contained in this CpaBufferList.
- `pSizeInBytes` – **[out]** 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.

Pre None.

Post None

1.4.3 Function cpaCyDhKeyGenPhase1

- Defined in file_api_lac_cpa_cy_dh.h

Function Documentation

CpaStatus `cpaCyDhKeyGenPhase1`(const *CpaInstanceHandle* instanceHandle, const *CpaCyGenFlatBufCbFunc* pDhPhase1Cb, void *pCallbackTag, const *CpaCyDhPhase1KeyGenOpData* *pPhase1KeyGenData, *CpaFlatBuffer* *pLocalOctetStringPV)

Function to implement Diffie-Hellman phase 1 operations.

See also:

CpaCyGenFlatBufCbFunc, *CpaCyDhPhase1KeyGenOpData*

Description: 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 size 6144 from RFC 3536 is 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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pDhPhase1Cb` – **[in]** 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.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback
- `pPhase1KeyGenData` – **[in]** 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.
- `pLocalOctetStringPV` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.4 Function cpaCyDhKeyGenPhase2Secret

- Defined in file_api_lac_cpa_cy_dh.h

Function Documentation

CpaStatus `cpaCyDhKeyGenPhase2Secret`(const *CpaInstanceHandle* instanceHandle, const *CpaCyGenFlatBufCbFunc* pDhPhase2Cb, void *pCallbackTag, const *CpaCyDhPhase2SecretKeyGenOpData* *pPhase2SecretKeyGenData, *CpaFlatBuffer* *pOctetStringSecretKey)

Function to implement Diffie-Hellman phase 2 operations.

See also:

CpaCyGenFlatBufCbFunc, *CpaCyDhPhase2SecretKeyGenOpData*

Description: 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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.

- `pDhPhase2Cb` – **[in]** 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.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.
- `pPhase2SecretKeyGenData` – **[in]** 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.
- `pOctetStringSecretKey` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.5 Function cpaCyDhQueryStats

- Defined in file_api_lac_cpa_cy_dh.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function “cpaCyDhQueryStats” with arguments “(const CpaInstanceHandle, struct _CpaCyDhStats*)”. Candidate function could not be parsed. Parsing error is Error when parsing function declaration. If the function has no return type: Error in declarator or parameters-and-qualifiers Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 10] CpaStatus CPA_DEPRECATED cpaCyDhQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyDhStats *pDhStats) ———-^ If the function has a return type: Error in declarator or parameters-and-qualifiers If pointer to member declarator: Invalid C++ declaration: Expected '::' in pointer to member (function). [error at 25] CpaStatus CPA_DEPRECATED cpaCyDhQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyDhStats *pDhStats) ——————————-^ If declarator-id: Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 25] CpaStatus CPA_DEPRECATED cpaCyDhQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyDhStats *pDhStats) ——————————-^

1.4.6 Function cpaCyDhQueryStats64

- Defined in file_api_lac_cpa_cy_dh.h

Function Documentation

CpaStatus `cpaCyDhQueryStats64`(const *CpaInstanceHandle* instanceHandle, *CpaCyDhStats64* *pDhStats)

Query statistics (64-bit version) for Diffie-Hellman operations

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.

Description: 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.

See also:

CpaCyDhStats64

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pDhStats` – **[out]** 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.

Pre Component has been initialized.

Post None

1.4.7 Function cpaCyDsaGenGParam

- Defined in file_api_lac_cpa_cy_dsa.h

Function Documentation

CpaStatus `cpaCyDsaGenGParam`(const *CpaInstanceHandle* instanceHandle, const *CpaCyDsaGenCbFunc* pCb, void *pCallbackTag, const *CpaCyDsaGParamGenOpData* *pOpData, *CpaBoolean* *pProtocolStatus, *CpaFlatBuffer* *pG)

Generate DSA G Parameter.

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.

Description: 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.
```

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.

See also:

CpaCyDsaGParamGenOpData, *CpaCyDsaGenCbFunc*

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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pProtocolStatus` – **[out]** The result passes/fails the DSA protocol related checks.
- `pG` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.8 Function cpaCyDsaGenPParam

- Defined in file_api_lac_cpa_cy_dsa.h

Function Documentation

CpaStatus `cpaCyDsaGenPParam`(const *CpaInstanceHandle* instanceHandle, const *CpaCyDsaGenCbFunc* pCb, void *pCallbackTag, const *CpaCyDsaPParamGenOpData* *pOpData, *CpaBoolean* *pProtocolStatus, *CpaFlatBuffer* *pP)

Generate DSA P Parameter.

See also:

CpaCyDsaPParamGenOpData, *CpaCyDsaGenCbFunc*

Description:

```
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.]

```

(continues on next page)

(continued from previous page)

```
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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pProtocolStatus` – **[out]** The result passes/fails the DSA protocol related checks.
- `pP` – **[out]** 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.

Pre The component has been initialized.

Post None

1.4.9 Function cpaCyDsaGenYParam

- Defined in file_api_lac_cpa_cy_dsa.h

Function Documentation

CpaStatus `cpaCyDsaGenYParam`(const *CpaInstanceHandle* instanceHandle, const *CpaCyDsaGenCbFunc* pCb, void *pCallbackTag, const *CpaCyDsaYParamGenOpData* *pOpData, *CpaBoolean* *pProtocolStatus, *CpaFlatBuffer* *pY)

Generate DSA Y Parameter.

See also:

CpaCyDsaYParamGenOpData, *CpaCyDsaGenCbFunc*

Description:

```
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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pProtocolStatus` – **[out]** The result passes/fails the DSA protocol related checks.
- `pY` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.10 Function cpaCyDsaQueryStats

- Defined in file_api_lac_cpa_cy_dsa.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "cpaCyDsaQueryStats" with arguments "(const CpaInstanceHandle, struct _CpaCyDsaStats*)". Candidate function could not be parsed. Parsing error is Error when parsing function declaration. If the function has no return type: Error in declarator or parameters-and-qualifiers Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 10] CpaStatus CPA_DEPRECATED cpaCyDsaQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyDsaStats *pDsaStats) ———-^ If the function has a return type: Error in declarator or parameters-and-qualifiers If pointer to member declarator: Invalid C++ declaration: Expected '::' in pointer to member (function). [error at 25] CpaStatus CPA_DEPRECATED cpaCyDsaQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyDsaStats *pDsaStats) ————————-^ If declarator-id: Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 25] CpaStatus CPA_DEPRECATED cpaCyDsaQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyDsaStats *pDsaStats) ————————-^

1.4.11 Function cpaCyDsaQueryStats64

- Defined in file_api_lac_cpa_cy_dsa.h

Function Documentation

CpaStatus `cpaCyDsaQueryStats64`(const *CpaInstanceHandle* instanceHandle, *CpaCyDsaStats64* *pDsaStats)

Query 64-bit statistics for a specific DSA instance.

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.

Description: 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.

See also:

CpaCyDsaStats

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pDsaStats` – **[out]** 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.

Pre Component has been initialized.

Post None

1.4.12 Function cpaCyDsaSignR

- Defined in file_api_lac_cpa_cy_dsa.h

Function Documentation

CpaStatus `cpaCyDsaSignR`(const *CpaInstanceHandle* instanceHandle, const *CpaCyDsaGenCbFunc* pCb, void *pCallbackTag, const *CpaCyDsaRSignOpData* *pOpData, *CpaBoolean* *pProtocolStatus, *CpaFlatBuffer* *pR)

Generate DSA R Signature.

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.

Description: This function generates the DSA R signature as described in FIPS 186-3 Section 4.6: r = (g^k mod p) mod q

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.

See also:

CpaCyDsaRSignOpData, *CpaCyDsaGenCbFunc*, *cpaCyDsaSignS()*, *cpaCyDsaSignRS()*

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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.

- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pProtocolStatus` – **[out]** The result passes/fails the DSA protocol related checks.
- `pR` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.13 Function cpaCyDsaSignRS

- Defined in file_api_lac_cpa_cy_dsa.h

Function Documentation

CpaStatus `cpaCyDsaSignRS(`const *CpaInstanceHandle* instanceHandle, const *CpaCyDsaRSSignCbFunc* pCb, void *pCallbackTag, const *CpaCyDsaRSSignOpData* *pOpData, *CpaBoolean* *pProtocolStatus, *CpaFlatBuffer* *pR, *CpaFlatBuffer* *pS`)`

Generate DSA R and S Signatures.

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.

Description: 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.

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)).

See also:

CpaCyDsaRSSignOpData, *CpaCyDsaRSSignCbFunc*, *cpaCyDsaSignR()*, *cpaCyDsaSignS()*

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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pProtocolStatus` – **[out]** The result passes/fails the DSA protocol related checks.
- `pR` – **[out]** DSA message signature r.

- `pS` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.14 Function cpaCyDsaSignS

- Defined in file_api_lac_cpa_cy_dsa.h

Function Documentation

CpaStatus `cpaCyDsaSignS(`const *CpaInstanceHandle* instanceHandle, const *CpaCyDsaGenCbFunc* pCb, void *pCallbackTag, const *CpaCyDsaSSignOpData* *pOpData, *CpaBoolean* *pProtocolStatus, *CpaFlatBuffer* *pS`)`

Generate DSA S Signature.

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.

Description: This function generates the DSA S signature as described in FIPS 186-3 Section 4.6:
s = (k^-1(z + xr)) 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 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.

See also:

CpaCyDsaSSignOpData, *CpaCyDsaGenCbFunc*, *cpaCyDsaSignR()*, *cpaCyDsaSignRS()*

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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pProtocolStatus` – **[out]** The result passes/fails the DSA protocol related checks.
- `pS` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.15 Function cpaCyDsaVerify

- Defined in file_api_lac_cpa_cy_dsa.h

Function Documentation

CpaStatus `cpaCyDsaVerify`(const *CpaInstanceHandle* instanceHandle, const *CpaCyDsaVerifyCbFunc* pCb, void *pCallbackTag, const *CpaCyDsaVerifyOpData* *pOpData, *CpaBoolean* *pVerifyStatus)

Verify DSA R and S signatures.

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.

Description: 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

A response status of ok (verifyStatus == CPA_TRUE) means v = r’. A response status of not ok (verifyStatus == CPA_FALSE) means v != r’.

See also:

CpaCyDsaVerifyOpData, *CpaCyDsaVerifyCbFunc*

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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pVerifyStatus` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.16 Function cpaCyEcdsaQueryStats64

- Defined in file_api_lac_cpa_cy_ecdsa.h

Function Documentation

CpaStatus `cpaCyEcdsaQueryStats64`(const *CpaInstanceHandle* instanceHandle, *CpaCyEcdsaStats64* *pEcdsaStats)

Query statistics for a specific ECDSA instance.

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.

Description: 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.

See also:

CpaCyEcdsaStats64

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pEcdsaStats` – **[out]** 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.

Pre Component has been initialized.

Post None

1.4.17 Function cpaCyEcdsaSignR

- Defined in file_api_lac_cpa_cy_ecdsa.h

Function Documentation

CpaStatus `cpaCyEcdsaSignR(`const *CpaInstanceHandle* instanceHandle, const *CpaCyEcdsaGenSignCbFunc* pCb, void *pCallbackTag, const *CpaCyEcdsaSignROpData* *pOpData, *CpaBoolean* *pSignStatus, *CpaFlatBuffer* *pR`)`

Generate ECDSA Signature R.

See also:

None

Description: 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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.

- `pSignStatus` – **[out]** In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
- `pR` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.18 Function cpaCyEcdsaSignRS

- Defined in file_api_lac_cpa_cy_ecdsa.h

Function Documentation

CpaStatus `cpaCyEcdsaSignRS`(const *CpaInstanceHandle* instanceHandle, const *CpaCyEcdsaSignRSCbFunc* pCb, void *pCallbackTag, const *CpaCyEcdsaSignRSOpData* *pOpData, *CpaBoolean* *pSignStatus, *CpaFlatBuffer* *pR, *CpaFlatBuffer* *pS)

Generate ECDSA Signature R & S.

See also:

None

Description: 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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pSignStatus` – **[out]** In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
- `pR` – **[out]** ECDSA message signature r.
- `pS` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.19 Function cpaCyEcdsaSignS

- Defined in file_api_lac_cpa_cy_ecdsa.h

Function Documentation

CpaStatus `cpaCyEcdsaSignS`(const *CpaInstanceHandle* instanceHandle, const *CpaCyEcdsaGenSignCbFunc* pCb, void *pCallbackTag, const *CpaCyEcdsaSignSOpData* *pOpData, *CpaBoolean* *pSignStatus, *CpaFlatBuffer* *pS)

Generate ECDSA Signature S.

See also:

None

Description: 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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.

- `pSignStatus` – **[out]** In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
- `pS` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.20 Function cpaCyEcdsaVerify

- Defined in file_api_lac_cpa_cy_ecdsa.h

Function Documentation

CpaStatus `cpaCyEcdsaVerify`(const *CpaInstanceHandle* instanceHandle, const *CpaCyEcdsaVerifyCbFunc* pCb, void *pCallbackTag, const *CpaCyEcdsaVerifyOpData* *pOpData, *CpaBoolean* *pVerifyStatus)

Verify ECDSA Public Key.

A response status of ok (verifyStatus == CPA_TRUE) means that the signature was verified

Description: This function performs ECDSA Verify as per ANSI X9.62 2005 section 7.4.

See also:

CpaCyEcdsaVerifyOpData, *CpaCyEcdsaVerifyCbFunc*

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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pVerifyStatus` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.21 Function cpaCyEcGenericPointMultiply

- Defined in file_api_lac_cpa_cy_ec.h

Function Documentation

CpaStatus `cpaCyEcGenericPointMultiply`(const *CpaInstanceHandle* instanceHandle, const *CpaCyEcPointMultiplyCbFunc* pCb, void *pCallbackTag, const *CpaCyEcGenericPointMultiplyOpData* *pOpData, *CpaBoolean* *pMultiplyStatus, *CpaFlatBuffer* *pXk, *CpaFlatBuffer* *pYk)

Generic ECC point multiplication operation.

See also:

CpaCyEcPointMultiplyOpData, *CpaCyEcPointMultiplyCbFunc* *CpaCyEcCurveType* *CpaCyEcCurveParameters*

Description: This is the generic ECC point multiplication operation, which is agnostic to the type of the curve used.

Context:

Assumptions: None

Side-Effects: None

Blocking: Yes when configured to operate in synchronous mode.

Reentrant: No

Thread-safe: Yes

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value, the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.

- `pOpData` – **[in]** 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.
- `pMultiplyStatus` – **[out]** In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
- `pXk` – **[out]** Pointer to xk flat buffer.
- `pYk` – **[out]** Pointer to yk flat buffer.

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.

Pre Component has been initialized.

Post None

1.4.22 Function cpaCyEcGenericPointVerify

- Defined in file_api_lac_cpa_cy_ec.h

Function Documentation

CpaStatus `cpaCyEcGenericPointVerify`(const *CpaInstanceHandle* instanceHandle, const *CpaCyEcPointVerifyCbFunc* pCb, void *pCallbackTag, const *CpaCyEcGenericPointVerifyOpData* *pOpData, *CpaBoolean* *pVerifyStatus)

Generic ECC point verification operation.

See also:

CpaCyEcGenericPointVerifyOpData, *CpaCyEcPointVerifyCbFunc* *CpaCyEcCurveType* *CpaCyEcCurveParameters*

Description: This is the generic ECC point verification operation, which is agnostic to the type of the curve used.

Context:

Assumptions: None

Side-Effects: None

Blocking: Yes when configured to operate in synchronous mode.

Reentrant: No

Thread-safe: Yes

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pVerifyStatus` – **[out]** In synchronous mode, the verification output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).

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.

Pre Component has been initialized.

Post None

1.4.23 Function cpaCyEcMontEdwdsPointMultiply

- Defined in file_api_lac_cpa_cy_ec.h

Function Documentation

CpaStatus `cpaCyEcMontEdwdsPointMultiply`(const *CpaInstanceHandle* instanceHandle, const *CpaCyEcPointMultiplyCbFunc* pCb, void *pCallbackTag, const *CpaCyEcMontEdwdsPointMultiplyOpData* *pOpData, *CpaBoolean* *pMultiplyStatus, *CpaFlatBuffer* *pXk, *CpaFlatBuffer* *pYk)

Perform EC Point Multiplication on an Edwards or Montgomery curve as defined in RFC#7748.

See also:

CpaCyEcMontEdwdsPointMultiplyOpData, CpaCyEcMontEdwdsPointMultiplyCbFunc

Description: This function performs Elliptic Curve Point Multiplication as per RFC#7748

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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.

- `pOpData` – **[in]** 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.
- `pMultiplyStatus` – **[out]** In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
- `pXk` – **[out]** Pointer to xk flat buffer.
- `pYk` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.24 Function cpaCyEcPointMultiply

- Defined in file_api_lac_cpa_cy_ec.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "cpaCyEcPointMultiply" with arguments "(const CpaInstanceHandle, const CpaCyEcPointMultiplyCbFunc, void*, const CpaCyEcPointMultiplyOpData*, CpaBoolean*, CpaFlatBuffer*, CpaFlatBuffer*)". Candidate function could not be parsed. Parsing error is Error when parsing function declaration. If the function has no return type: Error in declarator or parameters-and-qualifiers Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 10] CpaStatus CPA_DEPRECATED cpaCyEcPointMultiply (const CpaInstanceHandle instanceHandle, const CpaCyEcPointMultiplyCbFunc pCb, void *pCallbackTag, const CpaCyEcPointMultiplyOpData *pOpData, CpaBoolean *pMultiplyStatus, CpaFlatBuffer *pXk, CpaFlatBuffer *pYk) ———-^ If the function has a return type: Error in declarator or parameters-and-qualifiers If pointer to member declarator: Invalid C++ declaration: Expected '::' in pointer to member (function). [error at 25] CpaStatus CPA_DEPRECATED cpaCyEcPointMultiply (const CpaInstanceHandle instanceHandle, const CpaCyEcPointMultiplyCbFunc pCb, void *pCallbackTag, const CpaCyEcPointMultiplyOpData *pOpData, CpaBoolean *pMultiplyStatus, CpaFlatBuffer *pXk, CpaFlatBuffer *pYk) ———————–^ If declarator-id: Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 25] CpaStatus CPA_DEPRECATED cpaCyEcPointMultiply (const CpaInstanceHandle instanceHandle, const CpaCyEcPointMultiplyCbFunc pCb, void *pCallbackTag, const CpaCyEcPointMultiplyOpData *pOpData, CpaBoolean *pMultiplyStatus, CpaFlatBuffer *pXk, CpaFlatBuffer *pYk) ————————–^

1.4.25 Function cpaCyEcPointVerify

- Defined in file_api_lac_cpa_cy_ec.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "cpaCyEcPointVerify" with arguments "(const CpaInstanceHandle, const CpaCyEcPointVerifyCbFunc, void*, const CpaCyEcPointVerifyOpData*, CpaBoolean*)". Candidate function could not be parsed. Parsing error is Error when parsing function declaration. If the function has no return type: Error in declarator or parameters-and-qualifiers Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 10] CpaStatus CPA_DEPRECATED cpaCyEcPointVerify (const CpaInstanceHandle instanceHandle, const CpaCyEcPointVerifyCbFunc pCb, void *pCallbackTag, const CpaCyEcPointVerifyOpData *pOpData, CpaBoolean *pVerifyStatus) ———-^ If the function has a return type: Error in declarator or parameters-and-qualifiers If pointer to member declarator: Invalid C++ declaration: Expected '::' in pointer to member (function). [error at 25] CpaStatus CPA_DEPRECATED cpaCyEcPointVerify (const CpaInstanceHandle instanceHandle, const CpaCyEcPointVerifyCbFunc pCb, void *pCallbackTag, const CpaCyEcPointVerifyOpData *pOpData, CpaBoolean *pVerifyStatus) ————————–^ If declarator-id: Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 25] CpaStatus CPA_DEPRECATED cpaCyEcPointVerify (const CpaInstanceHandle instanceHandle, const CpaCyEcPointVerifyCbFunc pCb, void *pCallbackTag, const CpaCyEcPointVerifyOpData *pOpData, CpaBoolean *pVerifyStatus) ————————–^

1.4.26 Function cpaCyEcQueryStats64

- Defined in file_api_lac_cpa_cy_ec.h

Function Documentation

CpaStatus `cpaCyEcQueryStats64`(const *CpaInstanceHandle* instanceHandle, *CpaCyEcStats64* *pEcStats)

Query statistics for a specific EC instance.

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.

Description: 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.

See also:

CpaCyEcStats64

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pEcStats` – **[out]** 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.

Pre Component has been initialized.

Post None

1.4.27 Function cpaCyGetInstances

- Defined in file_api_lac_cpa_cy_common.h

Function Documentation

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

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

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.

Description: 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.

See also:

cpaCyGetNumInstances

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

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

Parameters

- `numInstances` – **[in]** 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.
- `cyInstances` – **[inout]** 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.

Pre None

Post None

1.4.28 Function cpaCyGetNumInstances

- Defined in file_api_lac_cpa_cy_common.h

Function Documentation

CpaStatus `cpaCyGetNumInstances(`*Cpa16U* *pNumInstances)

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

See also:

cpaCyGetInstances

Description: 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

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

Parameters `pNumInstances` – **[out]** 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.

Pre None

Post None

1.4.29 Function cpaCyGetStatusText

- Defined in file_api_lac_cpa_cy_common.h

Function Documentation

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

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

See also:

CpaStatus

Description: 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

Note: None

Parameters

- `instanceHandle` – **[in]** Handle to an instance of this API.
- `errStatus` – **[in]** The error condition that occurred
- `pStatusText` – **[out]** 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.

Pre None.

Post None

1.4.30 Function cpaCyInstanceGetInfo

- Defined in file_api_lac_cpa_cy_common.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "cpaCyInstanceGetInfo" with arguments "(const CpaInstanceHandle, struct _CpaInstanceInfo*)". Candidate function could not be parsed. Parsing error is Error when parsing function declaration. If the function has no return type: Error in declarator or parameters-and-qualifiers Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 10] CpaStatus CPA_DEPRECATED cpaCyInstanceGetInfo (const CpaInstanceHandle instanceHandle, struct _CpaInstanceInfo *pInstanceInfo) ———-^ If the function has a return type: Error in declarator or parameters-and-qualifiers If pointer to member declarator: Invalid C++ declaration: Expected '::' in pointer to member (function). [error at 25] CpaStatus CPA_DEPRECATED cpaCyInstanceGetInfo (const CpaInstanceHandle instanceHandle, struct

_CpaInstanceInfo *pInstanceInfo) ——————————-^ If declarator-id: Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 25] CpaStatus CPA_DEPRECATED cpaCyInstanceGetInfo (const CpaInstanceHandle instanceHandle, struct _CpaInstanceInfo *pInstanceInfo) ——————————-^

1.4.31 Function cpaCyInstanceGetInfo2

- Defined in file_api_lac_cpa_cy_common.h

Function Documentation

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

Function to get information on a particular instance.

See also:

cpaCyGetNumInstances, *cpaCyGetInstances*, CpaInstanceInfo

Description: 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

Note: None

Parameters

- `instanceHandle` – **[in]** Handle to an instance of this API to be initialized.
- `pInstanceInfo2` – **[out]** 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.

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

Post None

1.4.32 Function cpaCyInstanceSetNotificationCb

- Defined in file_api_lac_cpa_cy_common.h

Function Documentation

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

Subscribe for instance notifications.

See also:

CpaCyInstanceNotificationCbFunc

Description: 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

Note: None

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pInstanceNotificationCb` – **[in]** Instance notification callback function pointer.

- `pCallbackTag` – **[in]** 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.

Pre Instance has been initialized.

Post None

1.4.33 Function cpaCyKeyGenMgf

- Defined in file_api_lac_cpa_cy_key.h

Function Documentation

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

Mask Generation Function.

See also:

CpaCyKeyGenMgfOpData, *CpaCyGenFlatBufCbFunc*

Description: 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.

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

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*.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pKeyGenCb` – **[in]** 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.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.
- `pKeyGenMgfOpData` – **[in]** 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.
- `pGeneratedMaskBuffer` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.34 Function cpaCyKeyGenMgfExt

- Defined in file_api_lac_cpa_cy_key.h

Function Documentation

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

Extended Mask Generation Function.

See also:

CpaCyKeyGenMgfOpData, *CpaCyGenFlatBufCbFunc*

Description: 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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pKeyGenCb` – **[in]** 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.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.
- `pKeyGenMgfOpDataExt` – **[in]** 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.
- `pGeneratedMaskBuffer` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.35 Function cpaCyKeyGenQueryStats

- Defined in file_api_lac_cpa_cy_key.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "cpaCyKeyGenQueryStats" with arguments "(const CpaInstanceHandle, struct _CpaCyKeyGenStats*)". Candidate function could not be parsed. Parsing error is Error when parsing function declaration. If the function has no return type: Error in declarator or parameters-and-qualifiers Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 10] CpaStatus CPA_DEPRECATED cpaCyKeyGenQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyKeyGenStats *pKeyGenStats) ———-^ If the function has a return type: Error in declarator or parameters-and-qualifiers If pointer to member declarator: Invalid C++ declaration: Expected '::' in pointer to member (function). [error at 25] CpaStatus CPA_DEPRECATED cpaCyKeyGenQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyKeyGenStats *pKeyGenStats) ————————-^ If declarator-id: Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 25] CpaStatus CPA_DEPRECATED cpaCyKeyGenQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyKeyGenStats *pKeyGenStats) ————————-^

1.4.36 Function cpaCyKeyGenQueryStats64

- Defined in file_api_lac_cpa_cy_key.h

Function Documentation

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

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

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.

Description: 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.

See also:

CpaCyKeyGenStats64

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pKeyGenStats` – **[out]** 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.

Pre Component has been initialized.

Post None

1.4.37 Function cpaCyKeyGenSsl

- Defined in file_api_lac_cpa_cy_key.h

Function Documentation

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

SSL Key Generation Function.

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

See also:

CpaCyKeyGenSslOpData, *CpaCyGenFlatBufCbFunc*

Description: 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.

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

- `instanceHandle` – **[in]** Instance handle.
- `pKeyGenCb` – **[in]** 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.

- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.
- `pKeyGenSslOpData` – **[in]** 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.
- `pGeneratedKeyBuffer` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.38 Function cpaCyKeyGenTls

- Defined in file_api_lac_cpa_cy_key.h

Function Documentation

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

TLS Key Generation Function.

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

Description: 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).

See also:

CpaCyKeyGenTlsOpData, *CpaCyGenFlatBufCbFunc*

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

- `instanceHandle` – **[in]** Instance handle.
- `pKeyGenCb` – **[in]** 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.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.
- `pKeyGenTlsOpData` – **[in]** 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.
- `pGeneratedKeyBuffer` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.39 Function cpaCyKeyGenTls2

- Defined in file_api_lac_cpa_cy_key.h

Function Documentation

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

TLS Key Generation Function version 2.

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

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

See also:

CpaCyKeyGenTlsOpData, *CpaCyGenFlatBufCbFunc*

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

- `instanceHandle` – **[in]** Instance handle.
- `pKeyGenCb` – **[in]** 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.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.

- `pKeyGenTlsOpData` – **[in]** 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.
- `hashAlgorithm` – **[in]** Specifies the hash algorithm to use. According to RFC5246, this should be "SHA-256 or a stronger standard hash

  function."
- `pGeneratedKeyBuffer` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.40 Function cpaCyKeyGenTls3

- Defined in file_api_lac_cpa_cy_key.h

Function Documentation

CpaStatus `cpaCyKeyGenTls3`(const *CpaInstanceHandle* instanceHandle, const *CpaCyGenFlatBufCbFunc* pKeyGenCb, void *pCallbackTag, const *CpaCyKeyGenHKDFOpData* *pKeyGenTlsOpData, *CpaCyKeyHKDFCipherSuite* cipherSuite, *CpaFlatBuffer* *pGeneratedKeyBuffer)

TLS Key Generation Function version 3.

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

Description: This function is used for TLS key generation. It implements the TLS HKDF (HMAC Key Derivation Function) as defined by RFC5689 (HKDF) and RFC8446 (TLS 1.3).

See also:

CpaCyGenFlatBufCbFunc CpaCyKeyGenHKDFOpData

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

- `instanceHandle` – **[in]** Instance handle.
- `pKeyGenCb` – **[in]** 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.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.
- `pKeyGenTlsOpData` – **[in]** 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. The memory must be pinned and contiguous, suitable for DMA operations.
- `hashAlgorithm` – **[in]** Specifies the hash algorithm to use. According to RFC5246, this should be "SHA-256 or a stronger standard hash

  function."
- `pGeneratedKeyBuffer` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.41 Function cpaCyKptDeleteKey

- Defined in file_api_lac_cpa_cy_kpt.h

Function Documentation

CpaStatus `cpaCyKptDeleteKey(`*CpaInstanceHandle* instanceHandle, *CpaCyKptHandle* keyHandle, *CpaCyKptKeyManagementStatus* *pKptStatus)

Perform KPT delete keys function according to key handle

File: cpa_cy_kpt.h

See also:

None

Description: Before closing a QAT session(instance), an application that has previously stored its wrapping key in a QAT device using the KPT framework executes this call to delete its wrapping key in the QAT device.

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

Note: None

Parameters

- `instanceHandle` – **[in]** QAT service instance handle.
- `keyHandle` – **[in]** A 64-bit handle value
- `pkptstatus` – **[out]** One of the status codes denoted in the enumerate type CpaCyKptKeyManagementStatus CPA_CY_KPT_SUCCESS Key Deleted successfully CPA_CY_KPT_SWK_FAIL_NOT_FOUND For any reason the input handle cannot be found. CPA_CY_KPT_FAILED Operation failed due to unspecified reason

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.

Pre Component has been initialized.

Post None

1.4.42 Function cpaCyKptEcdsaSignRS

- Defined in file_api_lac_cpa_cy_kpt.h

Function Documentation

CpaStatus `cpaCyKptEcdsaSignRS`(const *CpaInstanceHandle* instanceHandle, const *CpaCyEcdsaSignRSCbFunc* pCb, void *pCallbackTag, const *CpaCyKptEcdsaSignRSOpData* *pOpData, *CpaBoolean* *pSignStatus, *CpaFlatBuffer* *pR, *CpaFlatBuffer* *pS, *CpaCyKptUnwrapContext* *pKptUnwrapContext)

Generate ECDSA Signature R & S.

See also:

None

Description: This function is a variant 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

Note: By virtue of invoking the cpaCyKptEcdsaSignRS, the implementation understands CpaCyEcdsaSignRSOpData contains an encrypted private key that requires unwrapping. KptUnwrapContext contains a '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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pSignStatus` – **[out]** In synchronous mode, the multiply output is valid (CPA_TRUE) or the output is invalid (CPA_FALSE).
- `pR` – **[out]** ECDSA message signature r.
- `pS` – **[out]** ECDSA message signature s.
- `pKptUnwrapContext` – **[in]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.43 Function cpaCyKptLoadKey

- Defined in file_api_lac_cpa_cy_kpt.h

Function Documentation

CpaStatus `cpaCyKptLoadKey(`*CpaInstanceHandle* instanceHandle, *CpaCyKptLoadKey* *pSWK, *CpaCyKptHandle* *keyHandle, *CpaCyKptKeyManagementStatus* *pKptStatus`)`

Perform KPT key loading function.

File: cpa_cy_kpt.h

See also:

None

Description: This function is invoked by a QAT application to load an encrypted symmetric wrapping key.

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

Note: None

Parameters

- `instanceHandle` – **[in]** QAT service instance handle.
- `pSWK` – **[in]** Encrypted SWK
- `keyHandle` – **[out]** A 64-bit handle value created by KPT

- `pKptStatus` – **[out]** One of the status codes denoted in the enumerate type CpaCyKptKeyManagementStatus CPA_CY_KPT_SUCCESS Key Loaded successfully CPA_CY_KPT_LOADKEY_FAIL_QUOTA_EXCEEDED_PER_VFID SWK count exceeds the configured maxmium value per VFID CPA_CY_KPT_LOADKEY_FAIL_QUOTA_EXCEEDED_PER_PASID SWK count exceeds the configured maxmium value per PASID CPA_CY_KPT_LOADKEY_FAIL_QUOTA_EXCEEDED SWK count exceeds the configured maxmium value when not scoped to VFID or PASID CPA_CY_KPT_FAILED Operation failed due to unspecified reason

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` – KPT-2.0 is not supported.

Pre Component has been initialized.

Post None

1.4.44 Function cpaCyKptQueryDeviceCredentials

- Defined in file_api_lac_cpa_cy_kpt.h

Function Documentation

CpaStatus `cpaCyKptQueryDeviceCredentials`(const *CpaInstanceHandle* instanceHandle, *CpaCyKptValidationKey* *pDevCredential, *CpaCyKptKeyManagementStatus* *pKptStatus)

Query KPT's Per-Part public key(I_pu) and signature from QAT device

See also:

File: cpa_cy_kpt.h

Description: This function is to query RSA3K Per-Part public key and its PKCS#1 v2.2 SHA-384 signature from the QAT device.

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.

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pDevCredential` – **[out]** Device Per-Part public key
- `pKptStatus` – **[out]** One of the status codes denoted in the enumerate type CpaCyKptKeyManagementStatus CPA_CY_KPT_SUCCESS Device credentials retreived successfully CPA_CY_KPT_FAILED Operation failed

Return values

- `CPA_STATUS_SUCCESS` – Function executed successfully.
- `CPA_STATUS_INVALID_PARAM` – Invalid parameter passed in.
- `CPA_STATUS_FAIL` – Function failed. Suggested course of action is to shutdown and restart.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.
- `CPA_STATUS_RESTARTING` – API implementation is restarting. Resubmit the request.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.45 Function cpaCyKptQueryIssuingKeys

- Defined in file_api_lac_cpa_cy_kpt.h

Function Documentation

CpaStatus `cpaCyKptQueryIssuingKeys`(const *CpaInstanceHandle* instanceHandle, *CpaFlatBuffer* *pPublicX509IssueCert, *CpaCyKptKeyManagementStatus* *pKptStatus)

Discovery and Provisioning APIs for KPT

Query KPT's issuing public key(R_Pu) and signature from QAT driver.

See also:

File: cpa_cy_kpt.h

Description: This function is to query the RSA3K issuing key and its PKCS#1 v2.2 SHA-384 signature from the QAT driver.

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.

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pIssueCert` – **[out]** KPT-2.0 Issuing certificate in PEM format as defined in RFC#7468
- `pKptStatus` – **[out]** One of the status codes denoted in the enumerate type CpaCyKptKeyManagementStatus CPA_CY_KPT_SUCCESS Issuing key retrieved successfully CPA_CY_KPT_FAILED Operation failed

Return values

- `CPA_STATUS_SUCCESS` – Function executed successfully.
- `CPA_STATUS_INVALID_PARAM` – Invalid parameter passed in.
- `CPA_STATUS_FAIL` – Function failed. Suggested course of action is to shutdown and restart.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.
- `CPA_STATUS_RESTARTING` – API implementation is restarting. Resubmit the request.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.46 Function cpaCyKptRsaDecrypt

- Defined in file_api_lac_cpa_cy_kpt.h

Function Documentation

CpaStatus `cpaCyKptRsaDecrypt`(const *CpaInstanceHandle* instanceHandle, const *CpaCyGenFlatBufCbFunc* pRsaDecryptCb, void *pCallbackTag, const *CpaCyKptRsaDecryptOpData* *pDecryptOpData, *CpaFlatBuffer* *pOutputData, *CpaCyKptUnwrapContext* *pKptUnwrapContext)

Usage APIs for KPT

Perform KPT-2.0 mode RSA decrypt primitive operation on the input data.

File: cpa_cy_kpt.h

See also:

CpaCyKptRsaDecryptOpData, *CpaCyGenFlatBufCbFunc*,

Description: This function is a 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

Note: By virtue of invoking cpaSyKptRsaDecrypt, the implementation understands that pDecryptOpData contains an encrypted private key that requires unwrapping. KptUnwrapContext contains a ‘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 CpaCyKptRsaDecryptOpData is a concatenation of cipher text and hash tag. For optimal performance, data pointers SHOULD be 8-byte aligned.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pRsaDecryptCb` – **[in]** 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.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.
- `pDecryptOpData` – **[in]** 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.
- `pOutputData` – **[out]** 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.
- `pKptUnwrapContext` – **[in]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.47 Function cpaCyLnModExp

- Defined in file_api_lac_cpa_cy_ln.h

Function Documentation

CpaStatus `cpaCyLnModExp`(const *CpaInstanceHandle* instanceHandle, const *CpaCyGenFlatBufCbFunc* pLnModExpCb, void *pCallbackTag, const *CpaCyLnModExpOpData* *pLnModExpOpData, *CpaFlatBuffer* *pResult)

Perform modular exponentiation operation.

result = (base ^ exponent) mod modulus

Description: This function performs modular exponentiation. It computes the following result based on the inputs:

See also:

CpaCyLnModExpOpData, *CpaCyGenFlatBufCbFunc*

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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pLnModExpCb` – **[in]** Pointer to callback function to be invoked when the operation is complete.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.

- `pLnModExpOpData` – **[in]** 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.
- `pResult` – **[out]** 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.

Pre The component has been initialized.

Post None

1.4.48 Function cpaCyLnModInv

- Defined in file_api_lac_cpa_cy_ln.h

Function Documentation

CpaStatus `cpaCyLnModInv`(const *CpaInstanceHandle* instanceHandle, const *CpaCyGenFlatBufCbFunc* pLnModInvCb, void *pCallbackTag, const *CpaCyLnModInvOpData* *pLnModInvOpData, *CpaFlatBuffer* *pResult)

Perform modular inversion operation.

result = (1/A) mod B.

Description: This function performs modular inversion. It computes the following result based on the inputs:

See also:

CpaCyLnModInvOpData, *CpaCyGenFlatBufCbFunc*

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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pLnModInvCb` – **[in]** Pointer to callback function to be invoked when the operation is complete.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.
- `pLnModInvOpData` – **[in]** 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.
- `pResult` – **[out]** 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.

Pre The component has been initialized.

Post None

1.4.49 Function cpaCyLnStatsQuery

- Defined in file_api_lac_cpa_cy_ln.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "cpaCyLnStatsQuery" with arguments "(const CpaInstanceHandle, struct _CpaCyLnStats*)". Candidate function could not be parsed. Parsing error is Error when parsing function declaration. If the function has no return type: Error in declarator or parameters-and-qualifiers Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 10] CpaStatus CPA_DEPRECATED cpaCyLnStatsQuery (const CpaInstance-Handle instanceHandle, struct _CpaCyLnStats *pLnStats) ———-^ If the function has a return type: Error in declarator or parameters-and-qualifiers If pointer to member declarator: Invalid C++ declaration: Expected '::' in pointer to member (function). [error at 25] CpaStatus CPA_DEPRECATED cpaCyLnStatsQuery (const CpaInstanceHandle instanceHandle, struct _CpaCyLnStats *pLnStats) ————————-^ If declarator-id: Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 25] CpaStatus CPA_DEPRECATED cpaCyLnStatsQuery (const CpaInstanceHandle instanceHandle, struct _CpaCyLnStats *pLnStats) ————————-^

1.4.50 Function cpaCyLnStatsQuery64

- Defined in file_api_lac_cpa_cy_ln.h

Function Documentation

CpaStatus `cpaCyLnStatsQuery64`(const *CpaInstanceHandle* instanceHandle, *CpaCyLnStats64* *pLnStats)

Query statistics (64-bit version) for large number operations

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.

Description: 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.

See also:

CpaCyLnStats

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pLnStats` – **[out]** 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.

Pre Acceleration Services unit has been initialized.

Post None

1.4.51 Function cpaCyPrimeQueryStats

- Defined in file_api_lac_cpa_cy_prime.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "cpaCyPrimeQueryStats" with arguments "(const CpaInstanceHandle, struct _CpaCyPrimeStats*)". Candidate function could not be parsed. Parsing error is Error when parsing function declaration. If the function has no return type: Error in declarator or parameters-and-qualifiers Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 10] CpaStatus CPA_DEPRECATED cpaCyPrimeQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyPrimeStats *pPrimeStats) ———-^ If the function has a return type: Error in declarator or parameters-and-qualifiers If pointer to member declarator: Invalid C++ declaration: Expected '::' in pointer to member (function). [error at 25] CpaStatus CPA_DEPRECATED cpaCyPrimeQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyPrimeStats *pPrimeStats) ————————-^ If declarator-id: Invalid C++ declaration: Expecting "(" in parameters-and-qualifiers. [error at 25] CpaStatus CPA_DEPRECATED cpaCyPrimeQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyPrimeStats *pPrimeStats) ————————-^

1.4.52 Function cpaCyPrimeQueryStats64

- Defined in file_api_lac_cpa_cy_prime.h

Function Documentation

CpaStatus `cpaCyPrimeQueryStats64`(const *CpaInstanceHandle* instanceHandle, *CpaCyPrimeStats64* *pPrimeStats)

1.4.53 Function cpaCyPrimeTest

- Defined in file_api_lac_cpa_cy_prime.h

Function Documentation

CpaStatus `cpaCyPrimeTest`(const *CpaInstanceHandle* instanceHandle, const *CpaCyPrimeTestCbFunc* pCb, void *pCallbackTag, const *CpaCyPrimeTestOpData* *pOpData, *CpaBoolean* *pTestPassed)

Prime Number Test Function.

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);

Description: 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.

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).

See also:

CpaCyPrimeTestOpData, *CpaCyPrimeTestCbFunc*

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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCb` – **[in]** Callback function pointer. If this is set to a NULL value the function will operate synchronously.
- `pCallbackTag` – **[in]** User-supplied value to help identify request.
- `pOpData` – **[in]** 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.
- `pTestPassed` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.54 Function cpaCyQueryCapabilities

- Defined in file_api_lac_cpa_cy_im.h

Function Documentation

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

Returns capabilities of a Cryptographic API instance

Description: 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

- `instanceHandle` – **[in]** Handle to an instance of this API.
- `pCapInfo` – **[out]** 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.

Pre The instance has been initialized via the *cpaCyStartInstance* function.

Post None

1.4.55 Function cpaCyRsaDecrypt

- Defined in file_api_lac_cpa_cy_rsa.h

Function Documentation

CpaStatus `cpaCyRsaDecrypt`(const *CpaInstanceHandle* instanceHandle, const *CpaCyGenFlatBufCbFunc* pRsaDecryptCb, void *pCallbackTag, const *CpaCyRsaDecryptOpData* *pDecryptOpData, *CpaFlatBuffer* *pOutputData)

Perform the RSA decrypt (or sign) primitive operation on the input data.

See also:

CpaCyRsaDecryptOpData, *CpaCyGenFlatBufCbFunc*, *cpaCyRsaGenKey()*, *cpaCyRsaEncrypt()*

Description: 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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pRsaDecryptCb` – **[in]** 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.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.
- `pDecryptOpData` – **[in]** 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.
- `pOutputData` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.56 Function cpaCyRsaEncrypt

- Defined in file_api_lac_cpa_cy_rsa.h

Function Documentation

CpaStatus `cpaCyRsaEncrypt`(const *CpaInstanceHandle* instanceHandle, const *CpaCyGenFlatBufCbFunc* pRsaEncryptCb, void *pCallbackTag, const *CpaCyRsaEncryptOpData* *pEncryptOpData, *CpaFlatBuffer* *pOutputData)

Perform the RSA encrypt (or verify) primitive operation on the input data.

See also:

CpaCyGenFlatBufCbFunc CpaCyRsaEncryptOpData cpaCyRsaGenKey() cpaCyRsaDecrypt()

Description: 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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pRsaEncryptCb` – **[in]** 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.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.
- `pEncryptOpData` – **[in]** 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.
- `pOutputData` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.57 Function cpaCyRsaGenKey

- Defined in file_api_lac_cpa_cy_rsa.h

Function Documentation

CpaStatus `cpaCyRsaGenKey`(const *CpaInstanceHandle* instanceHandle, const *CpaCyRsaKeyGenCbFunc* pRsaKeyGenCb, void *pCallbackTag, const *CpaCyRsaKeyGenOpData* *pKeyGenOpData, *CpaCyRsaPrivateKey* *pPrivateKey, *CpaCyRsaPublicKey* *pPublicKey)

Generate RSA keys.

See also:

CpaCyRsaKeyGenOpData, *CpaCyRsaKeyGenCbFunc*, *cpaCyRsaEncrypt()*, *cpaCyRsaDecrypt()*

Description: 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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pRsaKeyGenCb` – **[in]** 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.
- `pCallbackTag` – **[in]** Opaque User Data for this specific call. Will be returned unchanged in the callback.

- `pKeyGenOpData` – **[in]** 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.
- `pPrivateKey` – **[out]** 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.
- `pPublicKey` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.58 Function cpaCyRsaQueryStats

- Defined in file_api_lac_cpa_cy_rsa.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function “cpaCyRsaQueryStats” with arguments “(const CpaInstanceHandle, struct _CpaCyRsaStats*)”. Candidate function could not be parsed. Parsing error is Error when parsing function declaration. If the function has no return type: Error in declarator or parameters-and-qualifiers Invalid C++ declaration: Expecting “(” in parameters-and-qualifiers. [error at 10] CpaStatus CPA_DEPRECATED cpaCyRsaQueryStats (const CpaInstance-Handle instanceHandle, struct _CpaCyRsaStats *pRsaStats) ———-^ If the function has a return type: Error in declarator or parameters-and-qualifiers If pointer to member declarator: Invalid C++ declaration: Expected ‘::’ in pointer to member (function). [error at 25] CpaStatus CPA_DEPRECATED cpaCyRsaQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyRsaStats *pRsaStats) ————————-^ If declarator-id: Invalid C++ declaration: Expecting “(” in parameters-and-qualifiers. [error at 25] CpaStatus CPA_DEPRECATED cpaCyRsaQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCyRsaStats *pRsaStats) ————————-^

1.4.59 Function cpaCyRsaQueryStats64

- Defined in file_api_lac_cpa_cy_rsa.h

Function Documentation

CpaStatus `cpaCyRsaQueryStats64`(const *CpaInstanceHandle* instanceHandle, *CpaCyRsaStats64* *pRsaStats)

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

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.

Description: 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.

See also:

CpaCyRsaStats64

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pRsaStats` – **[out]** 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.

Pre Component has been initialized.

Post None

1.4.60 Function cpaCySetAddressTranslation

- Defined in file_api_lac_cpa_cy_im.h

Function Documentation

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

Sets the address translation function

See also:

None

Description: 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

- `instanceHandle` – **[in]** Handle to an instance of this API.
- `virtual2Physical` – **[in]** 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.

Pre None

Post None

1.4.61 Function cpaCyStartInstance

- Defined in file_api_lac_cpa_cy_im.h

Function Documentation

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

Cryptographic Component Initialization and Start function.

See also:

cpaCyStopInstance()

Description: 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

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

Parameters `instanceHandle` – **[out]** 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.

Pre None.

Post None

1.4.62 Function cpaCyStopInstance

- Defined in file_api_lac_cpa_cy_im.h

Function Documentation

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

Cryptographic Component Stop function.

See also:

cpaCyStartInstance()

Description: 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

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

Parameters `instanceHandle` – **[in]** 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.

Pre The component has been initialized via cpaCyStartInstance.

Post None

1.4.63 Function cpaCySymDpEnqueueOp

- Defined in file_api_lac_cpa_cy_sym_dp.h

Function Documentation

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

Enqueue a single symmetric cryptographic request.

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

Description: 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.

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.

See also:

cpaCySymDpInitSession, *cpaCySymDpPerformOpNow*

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

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.

Parameters

- `pOpData` – **[in]** 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.
- `performOpNow` – **[in]** 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.

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

Post None

1.4.64 Function cpaCySymDpEnqueueOpBatch

- Defined in file_api_lac_cpa_cy_sym_dp.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function “cpaCySymDpEnqueueOpBatch” with arguments (const Cpa32U, CpaCySymDpOpData*, const CpaBoolean) in doxygen xml output for project “qat_apiref_all” from directory: ./_doxygen/xml. Potential matches:

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

1.4.65 Function cpaCySymDpInitSession

- Defined in file_api_lac_cpa_cy_sym_dp.h

Function Documentation

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

Initialize a session for the symmetric cryptographic data plane API.

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

Description: 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.

The session can be removed using *cpaCySymDpRemoveSession*.

See also:

cpaCySymDpSessionCtxGetSize, *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

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

Parameters

- `instanceHandle` – **[in]** Instance to which the requests will be submitted.
- `pSessionSetupData` – **[in]** 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.
- `sessionCtx` – **[out]** 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.

Pre The component has been initialized.

Post None

1.4.66 Function cpaCySymDpPerformOpNow

- Defined in file_api_lac_cpa_cy_sym_dp.h

Function Documentation

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

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

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

Description: 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 also:

cpaCySymDpEnqueueOp, cpaCySymDpEnqueueOpBatch

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 `instanceHandle` – **[in]** 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.

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

Post None

1.4.67 Function cpaCySymDpRegCbFunc

- Defined in file_api_lac_cpa_cy_sym_dp.h

Function Documentation

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

Registration of the operation completion callback function.

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

Description: 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.

See also:

CpaCySymDpCbFunc

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

Note: None

Parameters

- `instanceHandle` – **[in]** Instance on which the callback function is to be registered.

- `pSymNewCb` – **[in]** 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.

Pre Component has been initialized.

Post None

1.4.68 Function cpaCySymDpRemoveSession

- Defined in file_api_lac_cpa_cy_sym_dp.h

Function Documentation

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

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

See also:

CpaCySymDpSessionCtx, *cpaCySymDpInitSession()*

Description: 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: No

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `sessionCtx` – **[inout]** 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.

Pre The component has been initialized.

Post None

1.4.69 Function cpaCySymDpSessionCtxGetDynamicSize

- Defined in file_api_lac_cpa_cy_sym_dp.h

Function Documentation

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

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

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.

Description: 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.

See also:

CpaCySymSessionSetupData cpaCySymDpSessionCtxGetSize() cpaCySymDpInitSession()

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pSessionSetupData` – **[in]** 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.
- `pSessionCtxSizeInBytes` – **[out]** 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.

Pre The component has been initialized.

Post None

1.4.70 Function cpaCySymDpSessionCtxGetSize

- Defined in file_api_lac_cpa_cy_sym_dp.h

Function Documentation

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

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

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.

Description: 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.

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.

See also:

CpaCySymSessionSetupData *cpaCySymDpSessionCtxGetDynamicSize()* *cpaCySymDpInitSession()*

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pSessionSetupData` – **[in]** 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.
- `pSessionCtxSizeInBytes` – **[out]** 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.

Pre The component has been initialized.

Post None

1.4.71 Function cpaCySymInitSession

- Defined in file_api_lac_cpa_cy_sym.h

Function Documentation

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

Initialize a session for symmetric cryptographic API.

See also:

CpaCySymSessionCtx, *CpaCySymCbFunc*, *CpaCySymSessionSetupData*, *cpaCySymRemoveSession()*, *cpaCySymPerformOp()*

Description: 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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pSymCb` – **[in]** Pointer to callback function to be registered. Set to NULL if the cpaCySymPerformOp function is required to work in a synchronous manner.
- `pSessionSetupData` – **[in]** 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.
- `sessionCtx` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.72 Function cpaCySymPerformOp

- Defined in file_api_lac_cpa_cy_sym.h

Function Documentation

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

Perform a symmetric cryptographic operation on an existing session.

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.

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

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 hash algorithm is NOT SHAKE
- The cipher algorithm is not SM4
- The cipher algorithm is not CPA_CY_SYM_CIPHER_CHACHA and the hash algorithm is not CPA_CY_SYM_HASH_POLY.
- The cipher algorithm is not CPA_CY_SYM_CIPHER_AES_GCM and the hash algorithm is not CPA_CY_SYM_HASH_AES_GCM.
- 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.

See also:

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

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

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.

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pCallbackTag` – **[in]** Opaque data that will be returned to the client in the callback.
- `pOpData` – **[in]** 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.

- `pSrcBuffer` – **[in]** 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.
- `pDstBuffer` – **[out]** 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.
- `pVerifyResult` – **[out]** 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.

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

Post None

1.4.73 Function cpaCySymQueryCapabilities

- Defined in file_api_lac_cpa_cy_sym.h

Function Documentation

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

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

Description: 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

- `instanceHandle` – **[in]** Handle to an instance of this API.
- `pCapInfo` – **[out]** 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.

Pre The instance has been initialized via the *cpaCyStartInstance* function.

Post None

1.4.74 Function cpaCySymQueryStats

- Defined in file_api_lac_cpa_cy_sym.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function “cpaCySymQueryStats” with arguments “(const CpaInstanceHandle, struct _CpaCySymStats*)”. Candidate function could not be parsed. Parsing error is Error when parsing function declaration. If the function has no return type: Error in declarator or parameters-and-qualifiers Invalid C++ declaration: Expecting “(” in parameters-and-qualifiers. [error at 10] CpaStatus CPA_DEPRECATED cpaCySymQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCySymStats *pSymStats) ———-^ If the function has a return type: Error in declarator or parameters-and-qualifiers If pointer to member declarator: Invalid C++ declaration: Expected ‘::’ in pointer to member (function). [error at 25] CpaStatus CPA_DEPRECATED cpaCySymQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCySymStats *pSymStats) ————————-^ If declarator-id: Invalid C++ declaration: Expecting “(” in parameters-and-qualifiers. [error at 25] CpaStatus CPA_DEPRECATED cpaCySymQueryStats (const CpaInstanceHandle instanceHandle, struct _CpaCySymStats *pSymStats) ————————-^

1.4.75 Function cpaCySymQueryStats64

- Defined in file_api_lac_cpa_cy_sym.h

Function Documentation

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

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

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.

Description: 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.

See also:

CpaCySymStats64

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pSymStats` – **[out]** 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.

Pre Component has been initialized.

Post None

1.4.76 Function cpaCySymRemoveSession

- Defined in file_api_lac_cpa_cy_sym.h

Function Documentation

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

Remove (delete) a symmetric cryptographic session.

See also:

CpaCySymSessionCtx, *cpaCySymInitSession()*

Description: 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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pSessionCtx` – **[inout]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.77 Function cpaCySymSessionCtxGetDynamicSize

- Defined in file_api_lac_cpa_cy_sym.h

Function Documentation

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

Gets the minimum size required to store a session context.

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 parameters provided. This size may be different for different setup data parameters.

Description: 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.

See also:

CpaCySymSessionSetupData *cpaCySymInitSession()* *cpaCySymSessionCtxGetSize()* *cpaCySymPerformOp()*

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pSessionSetupData` – **[in]** 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.
- `pSessionCtxSizeInBytes` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.78 Function cpaCySymSessionCtxGetSize

- Defined in file_api_lac_cpa_cy_sym.h

Function Documentation

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

Gets the size required to store a session context.

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.

Description: 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.

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.

See also:

CpaCySymSessionSetupData cpaCySymInitSession() cpaCySymSessionCtxGetDynamicSize() cpaCySymPerformOp()

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

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

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pSessionSetupData` – **[in]** 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.
- `pSessionCtxSizeInBytes` – **[out]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.79 Function cpaCySymSessionInUse

- Defined in file_api_lac_cpa_cy_sym.h

Function Documentation

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

Indicates whether there are outstanding requests on a given session.

Description: 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

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

1.4.80 Function cpaCySymUpdateSession

- Defined in file_api_lac_cpa_cy_sym.h

Function Documentation

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

Update a session.

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

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

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.

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

Parameters

- `sessionCtx` – **[in]** Identifies the session to be reset.
- `pSessionUpdateData` – **[in]** 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.

Pre The component has been initialized via cpaCyStartInstance function.

Post None

1.4.81 Function cpaDcBufferListGetMetaSize

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcBufferListGetMetaSize`(const *CpaInstanceHandle* instanceHandle, *Cpa32U* numBuffers, *Cpa32U* *pSizeInBytes)

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

See also:

cpaDcGetInstances()

Description: This function is used to obtain the size (in bytes) required to allocate a buffer descriptor for the pPrivateMetaData member in the CpaBufferList 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

Note: None

Parameters

- `instanceHandle` – **[in]** Handle to an instance of this API.
- `numBuffers` – **[in]** The number of pointers in the CpaBufferList. This is the maximum number of CpaFlatBuffers which may be contained in this CpaBufferList.
- `pSizeInBytes` – **[out]** 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.

Pre None

Post None

1.4.82 Function cpaDcChainGetSessionSize

- Defined in file_api_dc_cpa_dc_chain.h

Function Documentation

CpaStatus `cpaDcChainGetSessionSize(`*CpaInstanceHandle* dcInstance, *CpaDcChainOperations* operation, *Cpa8U* numSessions, *CpaDcChainSessionSetupData* *pSessionData, *Cpa32U* *pSessionSize)

Get the size of the memory required to hold the chaining sessions information.

See also:

cpaDcChainInitSession()

Description: The client of the Data Compression API is responsible for allocating sufficient memory to hold chaining sessions information. This function provides a way for determining the size of chaining sessions.

Context: No restrictions

Assumptions: None

Side-Effects: None

Blocking: No

Reentrant: No

Thread-safe: Yes

Note: Only a synchronous version of this function is provided.

Parameters

- `dcInstance` – **[in]** Instance handle.
- `operation` – **[in]** The operation for chaining
- `numSessions` – **[in]** Number of sessions for the chaining
- `pSessionData` – **[in]** Pointer to an array of CpaDcChainSessionSetupData structures. There should be numSessions entries in the array.
- `pSessionSize` – **[out]** On return, this parameter will be the size of the memory that will be required by *cpaDcChainInitSession()* for session data.

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.

Pre None

Post None

1.4.83 Function cpaDcChainGetSessionSize2

- Defined in file_api_dc_cpa_dc_chain.h

Function Documentation

CpaStatus `cpaDcChainGetSessionSize2(`*CpaInstanceHandle* dcInstance, *CpaDcChainOperations* operation, *Cpa8U* numSessions, *CpaDcChainSessionSetupData2* *pSessionData, *Cpa32U* *pSessionSize)

Get the size of the memory required to hold the chaining sessions information.

See also:

cpaDcChainInitSession2()

Description: The client of the Data Compression API is responsible for allocating sufficient memory to hold chaining sessions information. This function provides a way for determining the size of chaining sessions.

Context: No restrictions

Assumptions: None

Side-Effects: None

Blocking: No

Reentrant: No

Thread-safe: Yes

Note: Only a synchronous version of this function is provided. This function should be used when *cpaDcChainInitSession2()* is used.

Parameters

- `dcInstance` – **[in]** Instance handle.
- `operation` – **[in]** The operation for chaining
- `numSessions` – **[in]** Number of sessions for the chaining
- `pSessionData` – **[in]** Pointer to an array of CpaDcChainSessionSetupData2 structures with the additional session init parameters. There should be numSessions entries in the array.
- `pSessionSize` – **[out]** On return, this parameter will be the size of the memory that will be required by *cpaDcChainInitSession2()* for session data.

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.

Pre None

Post None

1.4.84 Function cpaDcChainInitSession

- Defined in file_api_dc_cpa_dc_chain.h

Function Documentation

CpaStatus `cpaDcChainInitSession(`*CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaDcChainOperations* operation, *Cpa8U* numSessions, *CpaDcChainSessionSetupData* *pSessionData, *CpaDcCallbackFn* callbackFn)

Initialize data compression chaining session

pSessionData Setup Rules

a. Each element in CpaDcChainSessionSetupData structure array provides (de)compression or a symmetric crypto session setup data.
b. The supported chaining operations are listed in CpaDcChainOperations. This enum indicates the number of operations in a chain and the order in which they are performed.
c. The order of entries in pSessionData[] should be consistent with the CpaDcChainOperations perform order. As an example, for CPA_DC_CHAIN_COMPRESS_THEN_ENCRYPT, pSessionData[0] holds the compression setup data and pSessionData[1] holds the encryption setup data..

d. The numSessions for each chaining operation are provided in the comments of enum CpaDcChainOperations.

e. For a (de)compression session, the corresponding pSessionData[]->sessType should be set to CPA_DC_CHAIN_COMPRESS_DECOMPRESS and pSessionData[]->pDcSetupData should point to a CpaDcSessionSetupData structure.

f. For a symmetric crypto session, the corresponding pSessionData[]->sessType should be set to CPA_DC_CHAIN_SYMMETRIC_CRYPTO and pSessionData[]->pCySetupData should point to a CpaCySymSessionSetupData structure.

g. Combined compression sessions are not supported for chaining.

h. Stateful compression is not supported for chaining.

i. Both CRC32 and Adler32 over the input data are supported for chaining.

Description: This function is used to initialize compression/decompression chaining sessions. This function returns a unique session handle each time this function is invoked. If the session has been configured with a callback function, then the order of the callbacks are guaranteed to be in the same order the compression or decompression requests were submitted for each session, so long as a single thread of execution is used for job submission. For data integrity computations the default CRC algorithm parameters are used.

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

See also:

None

Note: Only a synchronous version of this function is provided.

Parameters

- `dcInstance` – **[in]** Instance handle derived from discovery functions.
- `pSessionHandle` – **[inout]** Pointer to a session handle.
- `operation` – **[in]** The operations for chaining
- `numSessions` – **[in]** Number of sessions for chaining

- `pSessionData` – **[inout]** Pointer to an array of CpaDcChainSessionSetupData structures. There should be numSessions entries in the array.
- `callbackFn` – **[in]** For synchronous operation this callback shall be a null pointer.

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.

Pre dcInstance has been started using cpaDcStartInstance.

Post None

1.4.85 Function cpaDcChainInitSession2

- Defined in file_api_dc_cpa_dc_chain.h

Function Documentation

CpaStatus `cpaDcChainInitSession2`(*CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaDcChainOperations* operation, *Cpa8U* numSessions, *CpaDcChainSessionSetupData2* *pSessionData2, *CpaDcCallbackFn* callbackFn)

Initialize data compression chaining session with the ability to set the CRC algorithm parameters used by the data integrity CRCs.

pSessionData Setup Rules: The setup rules for *cpaDcChainInitSession()* function also applies to this function. When the session is initialized with this function, *cpaDcChainGetSessionSize2()* function should be used to query the size.

Description: This function is used to initialize compression/decompression chaining sessions with the ability to choose the CRC control data. This function returns a unique session handle each time this function is invoked. If the session has been configured with a callback function, then the order of the callbacks are guaranteed to be in the same order the compression or decompression requests were submitted for each session, so long as a single thread of execution is used for job submission.

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

See also:

cpaDcChainInitSession() cpaDcChainGetSessionSize2() CpaCrcControlData

---

Note: Only a synchronous version of this function is provided.

Parameters

- `dcInstance` – **[in]** Instance handle derived from discovery functions.
- `pSessionHandle` – **[inout]** Pointer to a session handle.
- `operation` – **[in]** The operations for chaining
- `numSessions` – **[in]** Number of sessions for chaining
- `pSessionData2` – **[inout]** Pointer to an array of CpaDcChainSessionSetupData2 structures. There should be numSessions entries in the array.
- `callbackFn` – **[in]** For synchronous operation this callback shall be a null pointer.

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.

Pre dcInstance has been started using cpaDcStartInstance.

Post None

1.4.86 Function cpaDcChainPerformOp

- Defined in file_api_dc_cpa_dc_chain.h

Function Documentation

CpaStatus `cpaDcChainPerformOp(`*CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaBufferList* *pSrcBuff, *CpaBufferList* *pDestBuff, *CpaDcChainOperations* operation, *Cpa8U* numOpDatas, *CpaDcChainOpData* *pChainOpData, *CpaDcChainRqResults* *pResults, void *callbackTag)

Submit a request to perform chaining operations.

pChainOpData Setup Rules

a. Each element in CpaDcChainOpData structure array holds either a (de)compression or a symmetric crypto operation data.

b. The order of entries in pChainOpData[] must be consistent with the order of operations described for the chaining operation in CpaDcChainOperations. As an example, for CPA_DC_CHAIN_HASH_THEN_COMPRESS, pChainOpData[0] must contain the hash operation data and pChainOpData[1] must contain the compress operation data.

c. The numOpDatas for each chaining operation are specified in the comments for the operation in CpaDcChainOperations.

d. For a (de)compression operation, the corresponding pChainOpData[]->opType should be set to CPA_DC_CHAIN_COMPRESS_DECOMPRESS and pChainOpData[]->pDcOp should point to a CpaDcOpData structure.

e. For a symmetric crypto operation, the corresponding pChainOpData[]->opType should be set to CPA_DC_CHAIN_SYMMETRIC_CRYPTO and pChainOpData[]->pCySymOp should point to a CpaCySymOpData structure.

   i. Partial packet processing is not supported.

   This function has identical buffer processing rules as *cpaDcCompressData()*.

   This function has identical checksum processing rules as *cpaDcCompressData()*, except:

   i. pResults->crc32 is available to application if CpaDcSessionSetupData->checksum is set to CPA_DC_CRC32

   ii. pResults->adler32 is available to application if CpaDcSessionSetupData->checksum is set to CPA_DC_ADLER32

   iii. Both pResults->crc32 and pResults->adler32 are available if CpaDcSessionSetupData->checksum is set to CPA_DC_CRC32_ADLER32

Description: This function is used to perform chaining operations over data from the source 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

Synchronous or asynchronous operation of the API is determined by the value of the callbackFn parameter passed to *cpaDcChainInitSession()* when the sessionHandle was setup. If a non-NULL value was specified then the supplied callback function will be invoked asynchronously with the response of this request.

This function has identical response ordering rules as *cpaDcCompressData()*.

See also:

cpaDcCompressData

Note: This function passes control to the compression service for chaining processing, the supported chaining operations are described in CpaDcChainOperations.

Parameters

- `dcInstance` – **[in]** Target service instance.
- `pSessionHandle` – **[inout]** Session handle.
- `pSrcBuff` – **[in]** Pointer to input data buffer.
- `pDestBuff` – **[out]** Pointer to output data buffer.
- `operation` – **[in]** Operation for the chaining request
- `numOpDatas` – **[in]** The entries size CpaDcChainOpData array
- `pChainOpData` – **[in]** Pointer to an array of CpaDcChainOpData structures. There should be numOpDatas entries in the array.
- `pResults` – **[inout]** Pointer to CpaDcChainRqResults
- `callbackTag` – **[in]** User supplied value to help correlate the callback with its associated request.

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_DC_BAD_DATA` – The input data was not properly formed.
- `CPA_STATUS_RESTARTING` – API implementation is restarting. Resubmit the request.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.

Pre pSessionHandle has been setup using *cpaDcChainInitSession()*

Post pSessionHandle has session related state information

1.4.87 Function cpaDcChainPerformOp2

- Defined in file_api_dc_cpa_dc_chain.h

Function Documentation

CpaStatus `cpaDcChainPerformOp2(`*CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaBufferList* *pSrcBuff, *CpaBufferList* *pDestBuff, *CpaBufferList* *pInterBuff, *CpaDcChainOpData2* opData, *CpaDcChainRqVResults* *pResults, void *callbackTag`)`

Submit a request to perform chaining with integrity operations.

User supplied opData contains pChainOpData

a. Refer to cpaDcChainPerformOp for pChainOpData Setup Rules

Description: This function is used to perform chaining operations over data from the source buffer with optional integrity checking.

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

This function has identical buffer processing rules as *cpaDcCompressData()*.

Synchronous or asynchronous operation of the API is determined by the value of the callbackFn parameter passed to *cpaDcChainInitSession()* when the sessionHandle was setup. If a non-NULL value was specified then the supplied callback function will be invoked asynchronously with the response of this request.

This function has identical response ordering rules as *cpaDcCompressData()*.

See also:

cpaDcCompressData

Note: This function passes control to the compression service for chaining processing, the supported chaining operations are described in CpaDcChainOperations.

Parameters

- `dcInstance` – **[in]** Target service instance.
- `pSessionHandle` – **[inout]** Session handle.
- `pSrcBuff` – **[in]** Pointer to input data buffer.
- `pDestBuff` – **[out]** Pointer to output data buffer.
- `pInterBuff` – **[in]** Pointer to intermediate buffer to be used

  as internal staging area for chaining operations.
- `opData` – **[in]** User supplied CpaDcChainOpData2 structure.
- `pResults` – **[inout]** Pointer to CpaDcChainRqVResults structure.
- `callbackTag` – **[in]** User supplied value to help correlate the callback with its associated request.

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_DC_BAD_DATA` – The input data was not properly formed.
- `CPA_STATUS_RESTARTING` – API implementation is restarting. Resubmit the request.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.

Pre pSessionHandle has been setup using *cpaDcChainInitSession()*

Post pSessionHandle has session related state information

1.4.88 Function cpaDcChainRemoveSession

- Defined in file_api_dc_cpa_dc_chain.h

Function Documentation

CpaStatus `cpaDcChainRemoveSession`(const *CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle)

Remove a compression chaining session.

See also:

cpaDcChainInitSession()

Description: This function will remove a previously initialized session handle 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 handle 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

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

Parameters

- `dcInstance` – **[in]** Instance handle.
- `pSessionHandle` – **[inout]** Session handle.

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.

Pre The component has been initialized via cpaDcStartInstance function.

Post None

1.4.89 Function cpaDcChainResetSession

- Defined in file_api_dc_cpa_dc_chain.h

Function Documentation

CpaStatus `cpaDcChainResetSession`(const *CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle)

Reset a compression chaining session.

See also:

cpaDcChainInitSession()

Description: This function will reset a previously initialized session handle. Reset will fail if outstanding calls still exist for the initialized session handle. The client needs to retry the reset function at a later time.

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

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

Parameters

- `dcInstance` – **[in]** Instance handle.
- `pSessionHandle` – **[inout]** Session handle.

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_UNSUPPORTED` – Function is not supported.

Pre The component has been initialized via cpaDcStartInstance function. The session has been initialized via cpaDcChainInitSession function.

Post None

1.4.90 Function cpaDcCompressData

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcCompressData(`*CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaBufferList* *pSrcBuff, *CpaBufferList* *pDestBuff, *CpaDcRqResults* *pResults, *CpaDcFlush* flushFlag, void *callbackTag)

Submit a request to compress a buffer of data.

In synchronous mode the function returns the error status returned from the service. In asynchronous mode the status is returned by the callback function.

Description: This API consumes data from the input buffer and generates compressed data in the output 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

This function may be called repetitively with input until all of the input has been consumed by the compression service and all the output has been produced.

When this function returns, it may be that all of the available data in the input buffer has not been compressed. This situation will occur when there is insufficient space in the output buffer. The calling application should note the amount of data processed, and clear the output buffer and then submit the request again, with the input buffer pointer to the data that was not previously compressed.

Relationship between input buffers and results buffers.

a. Implementations of this API must not modify the individual flat buffers of the input buffer list.

b. The implementation communicates the amount of data consumed from the source buffer list via pResults->consumed arg.

c. The implementation communicates the amount of data in the destination buffer list via pResults->produced arg.

Source Buffer Setup Rules

a. The buffer list must have the correct number of flat buffers. This is specified by the numBuffers element of the CpaBufferList.

b. Each flat buffer must have a pointer to contiguous memory that has been allocated by the calling application. The number of octets to be compressed or decompressed must be stored in the dataLenInBytes element of the flat buffer.

c. It is permissible to have one or more flat buffers with a zero length data store. This function will process all flat buffers until the destination buffer is full or all source data has been processed. If a buffer has zero length, then no data will be processed from that buffer.

Source Buffer Processing Rules.

a. The buffer list is processed in index order - SrcBuff->pBuffers[0] will be completely processed before SrcBuff->pBuffers[1] begins to be processed.

b. The application must drain the destination buffers. If the source data was not completely consumed, the application must resubmit the request.

c. On return, the pResults->consumed will indicate the number of bytes consumed from the input buffers.

Destination Buffer Setup Rules

a. The destination buffer list must have storage for processed data. This implies at least one flat buffer must exist in the buffer list.

b. For each flat buffer in the buffer list, the dataLenInBytes element must be set to the size of the buffer space.

c. It is permissible to have one or more flat buffers with a zero length data store. If a buffer has zero length, then no data will be added to that buffer.

Destination Buffer Processing Rules.

a. The buffer list is processed in index order - DestBuff->pBuffers[0] will be completely processed before DestBuff->pBuffers[1] begins to be processed.

b. On return, the pResults->produced will indicate the number of bytes written to the output buffers.

c. If processing has not been completed, the application must drain the destination buffers and resubmit the request. The application must reset the dataLenInBytes for each flat buffer in the destination buffer list.

Checksum rules. If a checksum is specified in the session setup data, then:

a. For the first request for a particular data segment the checksum is initialised internally by the implementation.

b. The checksum is maintained by the implementation between calls until the flushFlag is set to CPA_DC_FLUSH_FINAL indicating the end of a particular data segment.

    i. Intermediate checksum values are returned to the application, via the CpaDcRqResults structure, in response to each request. However these checksum values are not guaranteed to the valid until the call with flushFlag set to CPA_DC_FLUSH_FINAL completes successfully.

The application should set flushFlag to CPA_DC_FLUSH_FINAL to indicate processing a particular data segment is complete. It should be noted that this function may have to be called more than once to process data after the flushFlag parameter has been set to CPA_DC_FLUSH_FINAL if the destination buffer fills. Refer to buffer processing rules.

For stateful operations, when the function is invoked with flushFlag set to CPA_DC_FLUSH_NONE or CPA_DC_FLUSH_SYNC, indicating more data is yet to come, the function may or may not retain data. When the function is invoked with flushFlag set to CPA_DC_FLUSH_FULL or CPA_DC_FLUSH_FINAL, the function will process all buffered data.

For stateless operations, CPA_DC_FLUSH_FINAL will cause the BFINAL bit to be set for deflate compression. The initial checksum for the stateless operation should be set to 0. CPA_DC_FLUSH_NONE and CPA_DC_FLUSH_SYNC should not be used for stateless operations.

It is possible to maintain checksum and length information across *cpaDcCompressData()* calls with a stateless session without maintaining the full history state that is maintained by a stateful session. In this mode of operation, an initial checksum value of 0 is passed into the first *cpaDcCompressData()* call with the flush flag set to CPA_DC_FLUSH_FULL. On subsequent calls to *cpaDcCompressData()* for this session, the checksum passed to cpaDcCompressData should be set to the checksum value produced by the previous call to *cpaDcCompressData()*. When the last block of input data is passed to *cpaDcCompressData()*, the flush flag should be set to CPA_DC_FLUSH_FINAL. This will cause the BFINAL bit to be set in a deflate stream. It is the responsibility of the calling application to maintain overall lengths across the stateless requests and to pass the checksum produced by one request into the next request.

When an instance supports compressAndVerifyAndRecover, it is enabled by default when using *cpaDcCompressData()*. If this feature needs to be disabled, *cpaDcCompressData2()* must be used.

Synchronous or Asynchronous operation of the API is determined by the value of the callbackFn parameter passed to *cpaDcInitSession()* when the sessionHandle was setup. If a non-NULL value was specified then the supplied callback function will be invoked asynchronously with the response of this request.

Response ordering: For each session, the implementation must maintain the order of responses. That is, if in asynchronous mode, the order of the callback functions must match the order of jobs submitted by this function. In a simple synchronous mode implementation, the practice of submitting a request and blocking on its completion ensure ordering is preserved.

This limitation does not apply if the application employs multiple threads to service a single session.

If this API is invoked asynchronous, the return code represents the success or not of asynchronously scheduling the request. The results of the operation, along with the amount of data consumed and produced become available when the callback function is invoked. As such, pResults->consumed and pResults->produced are available only when the operation is complete.

The application must not use either the source or destination buffers until the callback has completed.

See also:

None

Note: This function passes control to the compression service for processing

Parameters

- `dcInstance` – **[in]** Target service instance.
- `pSessionHandle` – **[inout]** Session handle.
- `pSrcBuff` – **[in]** Pointer to data buffer for compression.
- `pDestBuff` – **[in]** Pointer to buffer space for data after compression.
- `pResults` – **[inout]** Pointer to results structure
- `flushFlag` – **[in]** Indicates the type of flush to be performed.
- `callbackTag` – **[in]** User supplied value to help correlate the callback with its associated request.

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_DC_BAD_DATA` – The input data was not properly formed.
- `CPA_STATUS_RESTARTING` – API implementation is restarting. Resubmit the request.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.

Pre pSessionHandle has been setup using *cpaDcInitSession()*

Post pSessionHandle has session related state information

1.4.91 Function cpaDcCompressData2

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcCompressData2(`*CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaBufferList* *pSrcBuff, *CpaBufferList* *pDestBuff, *CpaDcOpData* *pOpData, *CpaDcRqResults* *pResults, void *callbackTag`)`

Submit a request to compress a buffer of data.

See also:

cpaDcCompressData()

Description: This API consumes data from the input buffer and generates compressed data in the output buffer. This API is very similar to *cpaDcCompressData()* except it provides a CpaDcOpData structure for passing additional input parameters not covered in *cpaDcCompressData()*.

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

Note: This function passes control to the compression service for processing

Parameters

- `dcInstance` – **[in]** Target service instance.
- `pSessionHandle` – **[inout]** Session handle.
- `pSrcBuff` – **[in]** Pointer to data buffer for compression.
- `pDestBuff` – **[in]** Pointer to buffer space for data after compression.
- `pOpData` – **[inout]** Additional parameters.
- `pResults` – **[inout]** Pointer to results structure
- `callbackTag` – **[in]** User supplied value to help correlate the callback with its associated request.

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_DC_BAD_DATA` – The input data was not properly formed.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.
- `CPA_STATUS_RESTARTING` – API implementation is restarting. Resubmit the request.

Pre pSessionHandle has been setup using *cpaDcInitSession()*

Post pSessionHandle has session related state information

1.4.92 Function cpaDcDecompressData

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcDecompressData(`*CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaBufferList* *pSrcBuff, *CpaBufferList* *pDestBuff, *CpaDcRqResults* *pResults, *CpaDcFlush* flushFlag, void *callbackTag`)`

Submit a request to decompress a buffer of data.

This function may be called repetitively with input until all of the input has been provided and all the output has been consumed.

Description: This API consumes compressed data from the input buffer and generates uncompressed data in the output 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

This function has identical buffer processing rules as *cpaDcCompressData()*.

This function has identical checksum processing rules as *cpaDcCompressData()*.

The application should set flushFlag to CPA_DC_FLUSH_FINAL to indicate processing a particular compressed data segment is complete. It should be noted that this function may have to be called more than once to process data after flushFlag has been set if the destination buffer fills. Refer to buffer processing rules in *cpaDcCompressData()*.

Synchronous or Asynchronous operation of the API is determined by the value of the callbackFn parameter passed to *cpaDcInitSession()* when the sessionHandle was setup. If a non-NULL value was specified then the supplied callback function will be invoked asynchronously with the response of this request, along with the callbackTag specified in the function.

The same response ordering constraints identified in the cpaDcCompressData API apply to this function.

See also:

cpaDcCompressData()

Note: This function passes control to the compression service for decompression. The function returns the status from the service.

Parameters

- `dcInstance` – **[in]** Target service instance.
- `pSessionHandle` – **[inout]** Session handle.
- `pSrcBuff` – **[in]** Pointer to data buffer for compression.
- `pDestBuff` – **[in]** Pointer to buffer space for data after decompression.

- `pResults` – **[inout]** Pointer to results structure
- `flushFlag` – **[in]** When set to CPA_DC_FLUSH_FINAL, indicates that the input buffer contains all of the data for the compression session, allowing the function to release history data.
- `callbackTag` – **[in]** User supplied value to help correlate the callback with its associated request.

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_DC_BAD_DATA` – The input data was not properly formed.
- `CPA_STATUS_RESTARTING` – API implementation is restarting. Resubmit the request.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.

Pre pSessionHandle has been setup using *cpaDcInitSession()*

Post pSessionHandle has session related state information

1.4.93 Function cpaDcDecompressData2

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcDecompressData2`(*CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaBufferList* *pSrcBuff, *CpaBufferList* *pDestBuff, *CpaDcOpData* *pOpData, *CpaDcRqResults* *pResults, void *callbackTag)

Submit a request to decompress a buffer of data.

See also:

cpaDcDecompressData() cpaDcCompressData2() cpaDcCompressData()

Description: This API consumes compressed data from the input buffer and generates uncompressed data in the output buffer. This API is very similar to *cpaDcDecompressData()* except it provides a CpaDcOpData structure for passing additional input parameters not covered in *cpaDcDecompressData()*.

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

Note: This function passes control to the compression service for decompression. The function returns the status from the service.

Parameters

- `dcInstance` – **[in]** Target service instance.
- `pSessionHandle` – **[inout]** Session handle.
- `pSrcBuff` – **[in]** Pointer to data buffer for compression.
- `pDestBuff` – **[in]** Pointer to buffer space for data after decompression.
- `pOpData` – **[in]** Additional input parameters.
- `pResults` – **[inout]** Pointer to results structure
- `callbackTag` – **[in]** User supplied value to help correlate the callback with its associated request.

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_DC_BAD_DATA` – The input data was not properly formed.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.
- `CPA_STATUS_RESTARTING` – API implementation is restarting. Resubmit the request.

Pre pSessionHandle has been setup using *cpaDcInitSession()*

Post pSessionHandle has session related state information

1.4.94 Function cpaDcDeflateCompressBound

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcDeflateCompressBound`(const *CpaInstanceHandle* dcInstance, *CpaDcHuffType* huffType, *Cpa32U* inputSize, *Cpa32U* *outputSize)

Deflate Compression Bound API

See also:

None

Description: This function provides the maximum output buffer size for a Deflate compression operation in the "worst case" (non-compressible) scenario. It's primary purpose is for output buffer memory allocation.

Context: This is a synchronous function that will not 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

Note: This is a synchronous function and has no completion callback associated with it. The *cpaDcDeflateCompressBound()* API is intended to reduce the likelihood of overflow occurring during compression operations. An overflow may occur in some exception cases.

Parameters

- `dcInstance` – **[in]** Instance handle.
- `huffType` – **[in]** CpaDcHuffType to be used with this operation.
- `inputSize` – **[in]** Input Buffer size.
- `outputSize` – **[out]** Maximum output buffer size.

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.

Pre The component has been initialized via cpaDcStartInstance function.

Post None

1.4.95 Function cpaDcDpEnqueueOp

- Defined in file_api_dc_cpa_dc_dp.h

Function Documentation

CpaStatus `cpaDcDpEnqueueOp(`*CpaDcDpOpData* *pOpData, const *CpaBoolean* performOpNow`)`

Enqueue a single compression or decompression request.

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 *cpaDcDpRegCbFunc*) to be invoked. Callbacks within a session are guaranteed to be in the same order in which they were submitted.

Description: This function enqueues a single request to perform a compression, decompression operation.

The following restrictions apply to the pOpData parameter:

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

See also:

cpaDcDpPerformOpNow

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

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

Parameters

- `pOpData` – **[in]** 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 *cpaDcDpRegCbFunc*. See the above Description for some restrictions that apply to this parameter.
- `performOpNow` – **[in]** Flag to indicate 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 *cpaDcDpPerformOpNow*.

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.

Pre The session identified by pOpData->pSessionHandle was setup using *cpaDcDpInitSession* OR pOpData->pSetupData data structure was initialized for No-Session (Ns) usage. The instance identified by pOpData->dcInstance has had a callback function registered via *cpaDcDpRegCbFunc*.

Post None

1.4.96 Function cpaDcDpEnqueueOpBatch

- Defined in file_api_dc_cpa_dc_dp.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "cpaDcDpEnqueueOpBatch" with arguments (const Cpa32U, CpaDcDpOpData*, const CpaBoolean) in doxygen xml output for project "qat_apiref_all" from directory: ./_doxygen/xml. Potential matches:

```
- CpaStatus cpaDcDpEnqueueOpBatch(const Cpa32U numberRequests, CpaDcDpOpData *pOpData[],
↪const CpaBoolean performOpNow)
```

1.4.97 Function cpaDcDpGetSessionSize

- Defined in file_api_dc_cpa_dc_dp.h

Function Documentation

CpaStatus `cpaDcDpGetSessionSize`(*CpaInstanceHandle* dcInstance, *CpaDcSessionSetupData* *pSessionData, *Cpa32U* *pSessionSize)

Get the size of the memory required to hold the data plane session information.

Session data is expected to include interim checksum values, various counters and other other session related data that needs to persist between invocations. For a given implementation of this API, it is safe to assume that *cpaDcDpGetSessionSize()* will always return the same session 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

Description:

```
The client of the Data Compression API is responsible for
allocating sufficient memory to hold session information. This
function provides a means for determining the size of the session
information and statistics information.
```

Context: No restrictions

Assumptions: None

Side-Effects: None

Blocking: Yes

Reentrant: No

Thread-safe: Yes

See also:

cpaDcDpInitSession()

Note: Only a synchronous version of this function is provided.

Parameters

- `dcInstance` – **[in]** Instance handle.
- `pSessionData` – **[in]** Pointer to a user instantiated structure containing session data.
- `pSessionSize` – **[out]** On return, this parameter will be the size of the memory that will be required by *cpaDcInitSession()* for session data.

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.

Pre None

Post None

1.4.98 Function cpaDcDpInitSession

- Defined in file_api_dc_cpa_dc_dp.h

Function Documentation

CpaStatus `cpaDcDpInitSession(`*CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaDcSessionSetupData* *pSessionData)

Initialize compression or decompression data plane session.

This initializes opaque data structures in the session handle. Data compressed under this session will be compressed to the level specified in the pSessionData structure. Lower compression level numbers indicate a request for faster compression at the expense of compression ratio. Higher compression level numbers indicate a request for higher compression ratios at the expense of execution time.

Description: This function is used to initialize a compression/decompression session. A single session can be used for both compression and decompression requests. Clients MUST register a callback function for the compression service using this function. This function returns a unique session handle each time this function is invoked. The order of the callbacks are guaranteed to be in the same order the compression or decompression requests were submitted for each session, so long as a single thread of execution is used for job submission.

Context: This function may be called from any context.

Assumptions: None

Side-Effects: None

Blocking: Yes

Reentrant: No

Thread-safe: Yes

The session is opaque to the user application and the session handle contains job specific data.

The window size specified in the pSessionData must match exactly one of the supported window sizes specified in the capability structure. If a bi-directional session is being initialized, then the window size must be valid for both compress and decompress.

Note stateful sessions are not supported by this API.

See also:

None

Note: Only a synchronous version of this function is provided.

Parameters

- `dcInstance` – **[in]** Instance handle derived from discovery functions.
- `pSessionHandle` – **[inout]** Pointer to a session handle.
- `pSessionData` – **[inout]** Pointer to a user instantiated structure containing session data.

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.

Pre dcInstance has been started using *cpaDcStartInstance*.

Post None

1.4.99 Function cpaDcDpPerformOpNow

- Defined in file_api_dc_cpa_dc_dp.h

Function Documentation

CpaStatus `cpaDcDpPerformOpNow(`*CpaInstanceHandle* dcInstance`)`

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

See also:

cpaDcDpEnqueueOp, cpaDcDpEnqueueOpBatch

Description: This function triggers processing of previously enqueued requests on the referenced instance.

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 `dcInstance` – **[in]** 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.

Pre The component has been initialized via *cpaDcStartInstance* function. A compression session has been previously setup using the *cpaDcDpInitSession* function call.

Post None

1.4.100 Function cpaDcDpRegCbFunc

- Defined in file_api_dc_cpa_dc_dp.h

Function Documentation

CpaStatus `cpaDcDpRegCbFunc`(const *CpaInstanceHandle* dcInstance, const *CpaDcDpCallbackFn* pNewCb)

Registration of the operation completion callback function.

See also:

cpaDcDpCbFunc

Description: This function allows a completion callback function to be registered. The registered callback function is invoked on completion of asynchronous requests made via calls to *cpaDcDpEnqueueOp* or cpaDcDpEnqueueOpBatch.

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

Note: None

Parameters

- `dcInstance` – **[in]** Instance on which the callback function is to be registered.
- `pNewCb` – **[in]** Callback function for this instance.

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.

Pre Instance has been initialized.

Post None

1.4.101 Function cpaDcDpRemoveSession

- Defined in file_api_dc_cpa_dc_dp.h

Function Documentation

CpaStatus `cpaDcDpRemoveSession`(const *CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle)

Compression Data Plane Session Remove Function.

See also:

cpaDcDpInitSession

Description: This function will remove a previously initialized session handle 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 handle 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

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

Parameters

- `dcInstance` – **[in]** Instance handle.
- `pSessionHandle` – **[inout]** Session handle.

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.

Pre The component has been initialized via *cpaDcStartInstance* function.

Post None

1.4.102 Function cpaDcDpUpdateSession

- Defined in file_api_dc_cpa_dc_dp.h

Function Documentation

CpaStatus `cpaDcDpUpdateSession`(const *CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaDcSessionUpdateData* *pSessionUpdateData)

Compression Session Update Function.

See also:

cpaDcDpInitSession()

Description: This function is used to modify some select compression parameters of a previously initialized session handlei for a data plane session. Th update will fail if resources required for the new session settings are not available. Specifically, this function may fail if no intermediate buffers are associated with the instance, and the intended change would require these buffers. This function can be called at any time after a successful call of *cpaDcDpInitSession()*. This function does not change the parameters to compression request already in flight.

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: No

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

Parameters

- `dcInstance` – **[in]** Instance handle.
- `pSessionHandle` – **[inout]** Session handle.
- `pSessionUpdateData` – **[in]** Session Data.

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

Pre The component has been initialized via cpaDcStartInstance function. The session has been initialized via cpaDcDpInitSession function.

Post None

1.4.103 Function cpaDcGenerateFooter

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcGenerateFooter(`*CpaDcSessionHandle* pSessionHandle, *CpaFlatBuffer* *pDestBuff, *CpaDcRqResults* *pResults)

Generate compression footer.

For LZ4 compression, this function adds the LZ4 frame footer using XXH32 algorithm of the uncompressed data. The XXH32 checksum is added after the end mark. This section is defined in the documentation of the LZ4 frame format at: https://github.com/lz4/lz4/blob/dev/doc/lz4_Frame_format.md

Description: This function generates the footer for gzip, zlib or LZ4. The generated footer is stored it in the destination buffer. The type of footer created is determined using the compression algorithm selected for the session associated with the session handle.

Context: This function may be call from any context.

Assumptions: None

Side-Effects: All session variables are reset

Blocking: No

Reentrant: No

Thread-safe: Yes

An artifact of invoking this function for writing the footer data is that all opaque session specific data is re-initialized. If the compression level and file types are consistent, the upper level application can continue processing compression requests using the same session handle.

The produced element of the pResults structure will be incremented by the numbers bytes added to the buffer. The pointer to the buffer will not be modified. It is necessary for the application to ensure that there is always sufficient memory in the destination buffer to append the footer. In the event that the destination buffer would be too small to accept the footer, overflow will not be reported.

See also:

None

Note: Depending on the session variables, this function can add the alder32 footer to the zlib compressed data as defined in RFC1950. If required, it can also add the gzip footer, which is the crc32 of the uncompressed data and the length of the uncompressed data. This section is defined in RFC1952. The session variables used to determine the header type are CpaDcCompType and CpaDcChecksum, see cpaDcGenerateHeader for more details.

Parameters

- `pSessionHandle` – **[inout]** Session handle.
- `pDestBuff` – **[in]** Pointer to data buffer where the compression footer will go.
- `pResults` – **[inout]** Pointer to results structure filled by CpaDcCompressData. Updated with the results of this API call

Return values

- `CPA_STATUS_SUCCESS` – Function executed successfully.
- `CPA_STATUS_FAIL` – Function failed.

- `CPA_STATUS_UNSUPPORTED` – Function is not supported.
- `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.

Pre pSessionHandle has been setup using *cpaDcInitSession()* pResults structure has been filled by CpaDcCompressData().

1.4.104 Function cpaDcGenerateHeader

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcGenerateHeader(`*CpaDcSessionHandle* pSessionHandle, *CpaFlatBuffer* *pDestBuff, *Cpa32U* *count)

Generate compression header.

If the compression requires a gzip header, then this header requires at a minimum the following fields, defined in RFC1952: ID1: 0x1f ID2: 0x8b CM: Compression method = 8 for deflate

Description: This function generates the gzip header, zlib header or LZ4 frame header and stores it in the destination buffer. The type of header created is determined using the compression algorithm selected using *CpaDcSessionSetupData.compType*, for the session associated with the session handle.

Context: This function may be call from any context.

Assumptions: None

Side-Effects: None

Blocking: No

Reentrant: No

Thread-safe: Yes

The zlib header is defined in RFC1950 and this function must implement as a minimum: CM: four bit compression method - 8 is deflate with window size to 32k CINFO: four bit window size (see RFC1950 for details), 7 is 32k window FLG: defined as:

- Bits 0 - 4: check bits for CM, CINFO and FLG (see RFC1950)
- Bit 5: FDICT 0 = default, 1 is preset dictionary
- Bits 6 - 7: FLEVEL, compression level (see RFC 1950)

When LZ4 algorithm is used, this function can output a 7 byte frame header. This function will set the LZ4 frame header with:

- Magic number 0x184D2204
- The LZ4 max block size defined in the CpaDcSessionSetupData
- Flag byte as:
  - Version = 1
  - Block independence = 0
  - Block checksum = 0
  - Content size present = 0
  - Content checksum present = 1
  - Dictionary ID present = 0
- Content size = 0
- Dictionary ID = 0
- Header checksum = 1 byte representing the second byte of the XXH32 of the frame descriptor field.

The counter parameter will be set to the number of bytes added to the buffer. The pData will be not be changed.

For any of the compression algorithms used, the application is responsible to offset the pData pointer in CpaBufferList by the length of the header before calling the CpaDcCompressData() or CpaDcCompressData2() functions.

See also:

None

Note: When the deflate compression algorithm is used, this function can output a 10 byte gzip header or 2 byte zlib header to the destination buffer. The session properties are used to determine the header type. To output a Gzip or a Zlib header the session must have been initialized with CpaDcCompType CPA_DC_DEFLATE. To output a gzip header the session must have been initialized with CpaDcChecksum CPA_DC_CRC32. To output a zlib header the session must have been initialized with CpaDcChecksum CPA_DC_ADLER32. For CpaDcChecksum CPA_DC_NONE no header is output.

Parameters

- `pSessionHandle` – **[in]** Session handle.
- `pDestBuff` – **[in]** Pointer to data buffer where the compression header will go.
- `count` – **[out]** Pointer to counter filled in with header size.

Return values

- `CPA_STATUS_SUCCESS` – Function executed successfully.
- `CPA_STATUS_FAIL` – Function failed.
- `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.

Pre pSessionHandle has been setup using *cpaDcInitSession()*

1.4.105 Function cpaDcGetInstances

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcGetInstances(`*Cpa16U* numInstances, *CpaInstanceHandle* *dcInstances)

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

See also:

cpaDcGetInstances

Description:

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

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

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

Parameters

- `numInstances` – **[in]** Size of the array.
- `dcInstances` – **[out]** 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.

Pre None

Post None

1.4.106 Function cpaDcGetNumInstances

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcGetNumInstances(`*Cpa16U* *pNumInstances)

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

See also:

cpaDcGetInstances

Description:

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

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

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

---

Parameters `pNumInstances` – **[out]** 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.

Pre None

Post None

1.4.107 Function cpaDcGetNumIntermediateBuffers

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcGetNumIntermediateBuffers(`*CpaInstanceHandle* instanceHandle, *Cpa16U* *pNumBuffers)

Compression Component utility function to determine the number of intermediate buffers required by an implementation.

See also:

cpaDcStartInstance()

Description: This function will determine the number of intermediate buffer lists required by an implementation for a compression instance. These buffers should then be allocated and provided when calling *cpaDcStartInstance()* to start a compression instance that will use dynamic compression.

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

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

Parameters

- `instanceHandle` – **[inout]** Handle to an instance of this API to be initialized.
- `pNumBuffers` – **[out]** When the function returns, this will specify the number of buffer lists that should be used as intermediate buffers when calling *cpaDc-StartInstance()*.

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.

Pre None

Post None

1.4.108 Function cpaDcGetSessionSize

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcGetSessionSize(`*CpaInstanceHandle* dcInstance, *CpaDcSessionSetupData* *pSessionData, *Cpa32U* *pSessionSize, *Cpa32U* *pContextSize)

Get the size of the memory required to hold the session information.

It is expected that context data is comprised of the history and any data stores that are specific to the history such as linked lists or hash tables. For stateless sessions the context size returned from this function will be zero. For stateful sessions the context size returned will depend on the session setup data and may be zero.

Description:

```
The client of the Data Compression API is responsible for
allocating sufficient memory to hold session information and the context
data. This function provides a means for determining the size of the
session information and the size of the context data.
```

Context: No restrictions

Assumptions: None

Side-Effects: None

Blocking: No

Reentrant: No

Thread-safe: Yes

Session data is expected to include interim checksum values, various counters and other session related data that needs to persist between invocations. For a given implementation of this API, it is safe to assume that *cpaDcGetSessionSize()* will always return the same session 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.

See also:

cpaDcInitSession()

Note: Only a synchronous version of this function is provided.

Parameters

- `dcInstance` – **[in]** Instance handle.
- `pSessionData` – **[in]** Pointer to a user instantiated structure containing session data.
- `pSessionSize` – **[out]** On return, this parameter will be the size of the memory that will be required by *cpaDcInitSession()* for session data.
- `pContextSize` – **[out]** On return, this parameter will be the size of the memory that will be required for context data. Context data is save/restore data including history and any implementation specific data that is required for a save/restore operation.

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.

Pre None

Post None

1.4.109 Function cpaDcGetStats

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcGetStats(`*CpaInstanceHandle* dcInstance, *CpaDcStats* *pStatistics`)`

Retrieve statistics

See also:

None

Description: This API retrieves the current statistics for a compression instance.

Context: This function may be call from any context.

Assumptions: None

Side-Effects: None

Blocking: Yes

Reentrant: No

Thread-safe: Yes

Parameters

- `dcInstance` – **[in]** Instance handle.
- `pStatistics` – **[out]** Pointer to statistics structure.

Return values

- `CPA_STATUS_SUCCESS` – Function executed successfully.
- `CPA_STATUS_FAIL` – Function failed.
- `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.

Pre None

Post None

1.4.110 Function cpaDcGetStatusText

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcGetStatusText`(const *CpaInstanceHandle* dcInstance, const *CpaStatus* errStatus, *Cpa8S* *pStatusText)

Function to return a string indicating the specific error that occurred within the system.

See also:

CpaStatus

Description: When a function returns any error including CPA_STATUS_SUCCESS, the client can invoke this function to get a string which describes the general error condition, and if available additional information 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

Note: None

Parameters

- `dcInstance` – **[in]** Handle to an instance of this API.
- `errStatus` – **[in]** The error condition that occurred.
- `pStatusText` – **[inout]** Pointer to the string buffer that will be updated with the status text. The invoking application MUST allocate this buffer to be exactly 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 second time.
- `CPA_STATUS_INVALID_PARAM` – Invalid parameter passed in.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.

Pre None

Post None

1.4.111 Function cpaDcInitSession

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcInitSession`(*CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaDcSessionSetupData* *pSessionData, *CpaBufferList* *pContextBuffer, *CpaDcCallbackFn* callbackFn)

Initialize compression decompression session

This initializes opaque data structures in the session handle. Data compressed under this session will be compressed to the level specified in the pSessionData structure. Lower compression level numbers indicate a request for faster compression at the expense of compression ratio. Higher compression level numbers indicate a request for higher compression ratios at the expense of execution time.

Description: This function is used to initialize a compression/decompression session. This function specifies a BufferList for context data. A single session can be used for both compression and decompression requests. Clients MAY register a callback function for the compression service using this function. This function returns a unique session handle each time this function is invoked. If the session has been configured with a callback function, then the order of the callbacks are guaranteed to be in the same order the compression or decompression requests were submitted for each session, so long as a single thread of execution is used for job submission.

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

The session is opaque to the user application and the session handle contains job specific data.

The pointer to the ContextBuffer will be stored in session specific data if required by the implementation.

It is not permitted to have multiple outstanding asynchronous compression requests for stateful sessions. It is possible to add parallelization to compression by using multiple sessions.

The window size specified in the pSessionData must be match exactly one of the supported window sizes specified in the capabilities structure. If a bi-directional session is being initialized, then the window size must be valid for both compress and decompress.

See also:

None

Note: Only a synchronous version of this function is provided.

Parameters

- `dcInstance` – **[in]** Instance handle derived from discovery functions.
- `pSessionHandle` – **[inout]** Pointer to a session handle.
- `pSessionData` – **[inout]** Pointer to a user instantiated structure containing session data.
- `pContextBuffer` – **[in]** pointer to context buffer. This is not required for stateless operations. The total size of the buffer list must be equal to or larger than the specified contextSize retrieved from the *cpaDcGetSessionSize()* function.
- `callbackFn` – **[in]** For synchronous operation this callback shall be a null pointer.

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.

Pre dcInstance has been started using cpaDcStartInstance.

Post None

1.4.112 Function cpaDcInstanceGetInfo2

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcInstanceGetInfo2`(const *CpaInstanceHandle* instanceHandle, *CpaInstanceInfo2* *pInstanceInfo2)

Function to get information on a particular instance.

See also:

cpaDcGetNumInstances, *cpaDcGetInstances*, *CpaInstanceInfo2*

Description: This function will provide instance specific information through a *CpaInstanceInfo2* structure.

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

Assumptions: None

Side-Effects: None

Blocking: Yes

Reentrant: No

Thread-safe: Yes

Note: None

Parameters

- `instanceHandle` – **[in]** Handle to an instance of this API to be initialized.
- `pInstanceInfo2` – **[out]** 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.

Pre The client has retrieved an instanceHandle from successive calls to *cpaDcGetNumInstances* and *cpaDcGetInstances*.

Post None

1.4.113 Function cpaDcInstanceSetNotificationCb

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcInstanceSetNotificationCb`(const *CpaInstanceHandle* instanceHandle, const *CpaDcInstanceNotificationCbFunc* pInstanceNotificationCb, void *pCallbackTag)

Subscribe for instance notifications.

See also:

CpaDcInstanceNotificationCbFunc

Description: Clients of the CpaDc interface can subscribe for instance notifications by registering a *CpaDcInstanceNotificationCbFunc* function.

Context: This function may be called from any context.

Assumptions: None

Side-Effects: None

Blocking: No

Reentrant: No

Thread-safe: Yes

Note: None

Parameters

- `instanceHandle` – **[in]** Instance handle.
- `pInstanceNotificationCb` – **[in]** Instance notification callback function pointer.
- `pCallbackTag` – **[in]** 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.

Pre Instance has been initialized.

Post None

1.4.114 Function cpaDcLZ4CompressBound

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcLZ4CompressBound`(const *CpaInstanceHandle* dcInstance, *Cpa32U* inputSize, *Cpa32U* *outputSize)

LZ4 Compression Bound API

See also:

None

Description: This function provides the maximum output buffer size for a LZ4 compression operation in the "worst case" (non-compressible) scenario. It's primary purpose is for output buffer memory allocation.

Context: This is a synchronous function that will not 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

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

Parameters

- `dcInstance` – **[in]** Instance handle.

- `inputSize` – **[in]** Input Buffer size.
- `outputSize` – **[out]** Maximum output buffer size.

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.

Pre The component has been initialized via cpaDcStartInstance function.

Post None

1.4.115 Function cpaDcLZ4SCompressBound

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcLZ4SCompressBound`(const *CpaInstanceHandle* dcInstance, *Cpa32U* inputSize, *Cpa32U* *outputSize)

LZ4S Compression Bound API

See also:

None

Description: This function provides the maximum output buffer size for a LZ4S compression operation in the "worst case" (non-compressible) scenario. It's primary purpose is for output buffer memory allocation.

Context: This is a synchronous function that will not 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

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

Parameters

- `dcInstance` – **[in]** Instance handle.
- `inputSize` – **[in]** Input Buffer size.
- `outputSize` – **[out]** Maximum output buffer size.

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.

Pre The component has been initialized via cpaDcStartInstance function.

Post None

1.4.116 Function cpaDcNsCompressData

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcNsCompressData`(*CpaInstanceHandle* dcInstance, *CpaDcNsSetupData* *pSetupData, *CpaBufferList* *pSrcBuff, *CpaBufferList* *pDestBuff, *CpaDcOpData* *pOpData, *CpaDcRqResults* *pResults, *CpaDcCallbackFn* callbackFn, void *callbackTag)

Submit a request to compress a buffer of data without requiring a session to be created. This is a No-Session (Ns) variant of the cpaDcCompressData function.

Checksum rules. The checksum rules are the same as those for the session based APIs (cpaDc-CompressData or cpaDcCompressData2) with the following exception.

a. If the algorithm specified is CPA_DC_LZ4 or CPA_DC_LZ4S the xxHash32 checksum will not be maintained across calls to the API. The implication is that the xxHash32 value will only be valid for the output of a single request, no state will be saved. If an LZ4 frame is required, even in recoverable error scenarios such as CPA_DC_OVERFLOW, the checksum will not be continued. If that is required the session based API must be used.

Description: This API consumes data from the input buffer and generates compressed data in the output buffer. Unlike the other compression APIs this does not use a previously created session. This is a "one-shot" API that requests can be directly submitted to.

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

See also:

None

Note: This function passes control to the compression service for processing

Parameters

- `dcInstance` – **[in]** Target service instance.
- `pSetupData` – **[in]** Configuration structure for compression.
- `pSrcBuff` – **[in]** Pointer to data buffer for compression.
- `pDestBuff` – **[in]** Pointer to buffer space for data after compression.
- `pOpData` – **[in]** Additional input parameters.
- `pResults` – **[inout]** Pointer to results structure
- `callbackFn` – **[in]** For synchronous operation this callback shall be a null pointer.
- `callbackTag` – **[in]** User supplied value to help correlate the callback with its associated request.

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.

- `CPA_STATUS_RESTARTING` – API implementation is restarting. Resubmit the request.

Pre None

Post None

1.4.117 Function cpaDcNsDecompressData

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcNsDecompressData(`*CpaInstanceHandle* dcInstance, *CpaDcNsSetupData* *pSetupData, *CpaBufferList* *pSrcBuff, *CpaBufferList* *pDestBuff, *CpaDcOpData* *pOpData, *CpaDcRqResults* *pResults, *CpaDcCallbackFn* callbackFn, void *callbackTag`)`

Submit a request to decompress a buffer of data without requiring a session to be created. This is a No-Session (Ns) variant of the cpaDcDecompressData function.

See also:

cpaDcDecompressData() cpaDcCompressData2() cpaDcCompressData()

Description: This API consumes data from the input buffer and generates decompressed data in the output buffer. Unlike the other decompression APIs this does not use a previously created session. This is a "one-shot" API that requests can be directly submitted to.

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

Note: This function passes control to the decompression service. The function returns the status from the service.

Parameters

- `dcInstance` – **[in]** Target service instance.
- `pSetupData` – **[in]** Configuration structure for decompression..
- `pSrcBuff` – **[in]** Pointer to data buffer for decompression.
- `pDestBuff` – **[in]** Pointer to buffer space for data after decompression.
- `pOpData` – **[in]** Additional input parameters.
- `pResults` – **[inout]** Pointer to results structure
- `callbackFn` – **[in]** For synchronous operation this callback shall be a null pointer.
- `callbackTag` – **[in]** User supplied value to help correlate the callback with its associated request.

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.
- `CPA_STATUS_RESTARTING` – API implementation is restarting. Resubmit the request.

Pre None

Post None

1.4.118 Function cpaDcNsGenerateFooter

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcNsGenerateFooter(`*CpaDcNsSetupData* *pSetupData, *Cpa64U* totalLength, *CpaFlatBuffer* *pDestBuff, *CpaDcRqResults* *pResults`)`

Generate compression footer without requiring a session to be created. This is a No-Session (Ns) variant of the cpaDcGenerateFooter function.

To output an LZ4 footer the structure must have been initialized with with CpaDcCompType CPA_DC_LZ4. To output a gzip or zlib footer the structure must have been initialized with CpaDcCompType CPA_DC_DEFLATE. To output a gzip footer the structure must have been initialized with CpaDcChecksum CPA_DC_CRC32 and the totalLength parameter initialized to the total accumulated length of data processed. To output a zlib footer the structure must have been initialized with CpaDcChecksum CPA_DC_ADLER32. For CpaDcChecksum CPA_DC_NONE no footer is output.

Description: This API generates the footer for the required format and stores it in the destination buffer.

Context: This function may be call from any context.

Assumptions: None

Side-Effects: All session variables are reset

Blocking: No

Reentrant: No

Thread-safe: Yes

The produced element of the pResults structure will be incremented by the number of bytes added to the buffer. The pointer to the buffer will not be modified.

See also:

CpaDcNsSetupData *cpaDcNsGenerateHeader* *cpaDcGenerateFooter*

Note: This function outputs the required compression format footer to the destination buffer. The CpaDcNsSetupData structure fields are used to determine the footer type created.

Parameters

- `pSetupData` – **[in]** Pointer to Ns Configuration structure.
- `totalLength` – **[in]** Total accumulated length of input data processed. See description for formats that make use of this parameter.
- `pDestBuff` – **[in]** Pointer to data buffer where the compression footer will go.
- `pResults` – **[inout]** Pointer to results structure filled by CpaDcNsCompressData. Updated with the results of this API call

Return values

- `CPA_STATUS_SUCCESS` – Function executed successfully.
- `CPA_STATUS_FAIL` – Function failed.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.
- `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.

Pre pResults structure has been filled by CpaDcNsCompressData().

1.4.119 Function cpaDcNsGenerateHeader

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcNsGenerateHeader(`*CpaDcNsSetupData* *pSetupData, *CpaFlatBuffer* *pDestBuff, *Cpa32U* *count`)`

Generate compression header without requiring a session to be created. This is a No-Session (Ns) variant of the cpaDcGenerateHeader function.

To output an LZ4 header the structure must have been initialized with with CpaDcCompType CPA_DC_LZ4. To output a gzip or zlib header the structure must have been initialized with CpaDcCompType CPA_DC_DEFLATE. To output a gzip header the structure must have been initialized with CpaDcChecksum CPA_DC_CRC32. To output a zlib header the structure must have been initialized with CpaDcChecksum CPA_DC_ADLER32. For CpaDcChecksum CPA_DC_NONE no header is output.

Description: This API generates the required compression format header and stores it in the output buffer.

Context: This function may be called from any context.

Assumptions: None

Side-Effects: None

Blocking: No

Reentrant: No

Thread-safe: Yes

The counter parameter will be set to the number of bytes added to the buffer.

See also:

cpaDcGenerateHeader

Note: This function outputs the required compression format header to the destination buffer. The CpaDcNsSetupData structure fields are used to determine the header type.

Parameters

- `pSetupData` – **[in]** Pointer to Ns Configuration structure.
- `pDestBuff` – **[in]** Pointer to data buffer where the compression header will go.
- `count` – **[out]** Pointer to counter filled in with header size.

Return values

- `CPA_STATUS_SUCCESS` – Function executed successfully.
- `CPA_STATUS_FAIL` – Function failed.
- `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.

Pre None

1.4.120 Function cpaDcQueryCapabilities

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcQueryCapabilities(`*CpaInstanceHandle* dcInstance, *CpaDcInstanceCapabilities* *pInstanceCapabilities)

Retrieve Instance Capabilities

See also:

None

Description: This function is used to retrieve the capabilities matrix of an instance.

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

Assumptions: None

Side-Effects: None

Blocking: Yes

Reentrant: No

Thread-safe: Yes

Note: Only a synchronous version of this function is provided.

Parameters

- `dcInstance` – **[in]** Instance handle derived from discovery functions
- `pInstanceCapabilities` – **[inout]** Pointer to a capabilities struct

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.

Pre None

Post None

1.4.121 Function cpaDcRemoveSession

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcRemoveSession`(const *CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle)

Compression Session Remove Function.

See also:

cpaDcInitSession()

Description: This function will remove a previously initialized session handle 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 handle 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

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

Parameters

- `dcInstance` – **[in]** Instance handle.
- `pSessionHandle` – **[inout]** Session handle.

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.

Pre The component has been initialized via cpaDcStartInstance function.

Post None

1.4.122 Function cpaDcResetSession

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcResetSession`(const *CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle)

Compression Session Reset Function.

See also:

cpaDcInitSession()

Description: This function will reset a previously initialized session handle Reset will fail if outstanding calls still exist for the initialized session handle. The client needs to retry the reset function at a later time.

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

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

Parameters

- `dcInstance` – **[in]** Instance handle.
- `pSessionHandle` – **[inout]** Session handle.

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_UNSUPPORTED` – Function is not supported.

Pre The component has been initialized via cpaDcStartInstance function. The session has been initialized via cpaDcInitSession function.

Post None

1.4.123 Function cpaDcResetXXHashState

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcResetXXHashState`(const *CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle)

Reset of the xxHash internal state on a session.

See also:

Description: This function will reset the internal xxHash state maintained within a session. This would be used in conjunction with the *CpaDcSessionSetupData.accumulateXXHash* flag being set to TRUE for this session. It will enable resetting (reinitialising) just the xxHash calculation back to the state when the session was first initialised.

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

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

Parameters

- `dcInstance` – **[in]** Instance handle.
- `pSessionHandle` – **[inout]** Session handle.

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.

Pre The component has been initialized via cpaDcStartInstance function. The session has been initialized via cpaDcInitSession function.

Post None

1.4.124 Function cpaDcSetAddressTranslation

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcSetAddressTranslation`(const *CpaInstanceHandle* instanceHandle, *CpaVirtualToPhysical* virtual2Physical)

Set Address Translation function

See also:

None

Description: 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.

Assumptions: None

Side-Effects: None

Blocking: This function is synchronous and blocking.

Reentrant: No

Thread-safe: Yes

Parameters

- `instanceHandle` – **[in]** Data Compression API instance handle.
- `virtual2Physical` – **[in]** 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.

Pre None

Post None

1.4.125 Function cpaDcStartInstance

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcStartInstance(`*CpaInstanceHandle* instanceHandle, *Cpa16U* numBuffers, *CpaBufferList* **pIntermediateBuffers)

Compression Component Initialization and Start function.

If required by an implementation, this function can be provided with instance specific intermediate buffers. The intent is to provide an instance specific location to store intermediate results during dynamic instance Huffman tree compression requests. The memory should be accessible by the compression engine. The buffers are to support deflate compression with dynamic Huffman Trees. Each buffer list should be similar in size to twice the destination buffer size passed to the compress API. The number of intermediate buffer lists may vary between implementations and so *cpaDcGetNumIntermediateBuffers()* should be called first to determine the number of intermediate buffers required by the implementation.

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

If not required, this parameter can be passed in as NULL.

See also:

cpaDcStopInstance() cpaDcGetNumIntermediateBuffers()

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

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

Parameters

- `instanceHandle` – **[inout]** Handle to an instance of this API to be initialized.
- `numBuffers` – **[in]** Number of buffer lists represented by the pIntermediate-Buffers parameter. Note: *cpaDcGetNumIntermediateBuffers()* can be used to determine the number of intermediate buffers that an implementation requires.
- `pIntermediateBuffers` – **[in]** Optional pointer to Instance specific DRAM buffer.

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.

Pre None

Post None

1.4.126 Function cpaDcStopInstance

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcStopInstance(`*CpaInstanceHandle* instanceHandle`)`

Compress Component Stop function.

See also:

cpaDcStartInstance()

Description: This function will stop the Compression 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 cpaDcStartInstance 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

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

Parameters `instanceHandle` – **[in]** 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.

Pre The component has been initialized via cpaDcStartInstance

Post None

1.4.127 Function cpaDcUpdateSession

- Defined in file_api_dc_cpa_dc.h

Function Documentation

CpaStatus `cpaDcUpdateSession`(const *CpaInstanceHandle* dcInstance, *CpaDcSessionHandle* pSessionHandle, *CpaDcSessionUpdateData* *pSessionUpdateData)

Compression Session Update Function.

See also:

cpaDcInitSession()

Description: This function is used to modify some select compression parameters of a previously initialized session handle. Th update will fail if resources required for the new session settings are not available. Specifically, this function may fail if no intermediate buffers are associated with the instance, and the intended change would require these buffers. This function can be called at any time after a successful call of *cpaDcInitSession()*. This function does not change the parameters to compression request already in flight.

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

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

Parameters

- `dcInstance` – **[in]** Instance handle.
- `pSessionHandle` – **[inout]** Session handle.
- `pSessionUpdateData` – **[in]** Session Data.

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

Pre The component has been initialized via cpaDcStartInstance function. The session has been initialized via cpaDcInitSession function.

Post None

1.4.128 Function cpaFreeInstance

- Defined in file_api_cpa.h

Function Documentation

CpaStatus `cpaFreeInstance(`*CpaInstanceHandle* instanceHandle`)`

Free a service instance.

See also:

None

Description: This function is used to free a service instance.

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 `instanceHandle` – **[in]** Instance to free.

Return values

- `CPA_STATUS_SUCCESS` – An instance was successfully freed.
- `CPA_STATUS_FAIL` – Function failed to free an instance.
- `CPA_STATUS_INVALID_PARAM` – Invalid parameter passed in.
- `CPA_STATUS_UNSUPPORTED` – Function is not supported.

Pre None

Post None

1.4.129 Function cpaGetInstances

- Defined in file_api_cpa.h

Function Documentation

CpaStatus `cpaGetInstances`(const *CpaAccelerationServiceType* accelerationServiceType, *Cpa16U* numInstances, *CpaInstanceHandle* *cpaInstances)

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

File: cpa.h

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

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

See also:

cpaGetNumInstances

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

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

---

Parameters

- `accelerationServiceType` – **[in]** Acceleration Service requested
- `numInstances` – **[in]** Size of the array. If the value is greater than the number of instances supported, then an error (*CPA_STATUS_INVALID_PARAM*) is returned.
- `cpaInstances` – **[inout]** 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.

Pre None

Post None

1.4.130 Function cpaGetNumInstances

- Defined in file_api_cpa.h

Function Documentation

CpaStatus `cpaGetNumInstances`(const *CpaAccelerationServiceType* accelerationServiceType, *Cpa16U* *pNumInstances)

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

File: cpa.h

See also:

cpaGetInstances

Description: This function will get the number of instances that are supported for the required Acceleration Service by an implementation of the CPA API. This number is then used to determine the size of the array that must be passed to *cpaGetInstances()*.

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

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

---

Parameters

- `accelerationServiceType` – **[in]** Acceleration Service required

- `pNumInstances` – **[out]** 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.

Pre None

Post None

1.5 Defines

1.5.1 Define CPA_BITMAP

- Defined in file_api_cpa_types.h

Define Documentation

`CPA_BITMAP`(name, sizeInBits)

Declare a bitmap of specified size (in bits).

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

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

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.

1.5.2 Define CPA_BITMAP_BIT_CLEAR

- Defined in file_api_cpa_types.h

Define Documentation

`CPA_BITMAP_BIT_CLEAR`(bitmask, bit)

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

1.5.3 Define CPA_BITMAP_BIT_SET

- Defined in file_api_cpa_types.h

Define Documentation

`CPA_BITMAP_BIT_SET`(bitmask, bit)

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

File: cpa_types.h

1.5.4 Define CPA_BITMAP_BIT_TEST

- Defined in file_api_cpa_types.h

Define Documentation

`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).

1.5.5 Define CPA_CY_API_VERSION_AT_LEAST

- Defined in file_api_lac_cpa_cy_common.h

Define Documentation

`CPA_CY_API_VERSION_AT_LEAST`(major, minor)

CPA CY API version at least

File: cpa_cy_common.h

Description: The minimal supported CPA_CY API version. Allow to check if the API version is equal or above some version to avoid compilation issues with an older API version.

1.5.6 Define CPA_CY_API_VERSION_LESS_THAN

- Defined in file_api_lac_cpa_cy_common.h

Define Documentation

`CPA_CY_API_VERSION_LESS_THAN`(major, minor)

CPA CY API version less than

File: cpa_cy_common.h

Description: The maximum supported CPA_CY API version. Allow to check if the API version is below some version to avoid compilation issues with a newer API version.

1.5.7 Define CPA_CY_API_VERSION_NUM_MAJOR

- Defined in file_api_lac_cpa_cy_common.h

Define Documentation

`CPA_CY_API_VERSION_NUM_MAJOR`

CPA CY Major Version Number

Description: The CPA_CY API major version number. This number will be incremented when significant churn to the API has occurred. The combination of the major and minor number definitions represent the complete version number for this interface.

1.5.8 Define CPA_CY_API_VERSION_NUM_MINOR

- Defined in file_api_lac_cpa_cy_common.h

Define Documentation

`CPA_CY_API_VERSION_NUM_MINOR`

CPA CY Minor Version Number

Description: The CPA_CY API minor version number. This number will be incremented when minor changes to the API has occurred. The combination of the major and minor number definitions represent the complete version number for this interface.

1.5.9 Define CPA_CY_HKDF_KEY_MAX_HMAC_SZ

- Defined in file_api_lac_cpa_cy_key.h

Define Documentation

`CPA_CY_HKDF_KEY_MAX_HMAC_SZ`

space in bytes PSK or (EC)DH

1.5.10 Define CPA_CY_HKDF_KEY_MAX_INFO_SZ

- Defined in file_api_lac_cpa_cy_key.h

Define Documentation

`CPA_CY_HKDF_KEY_MAX_INFO_SZ`

space in bytes of CPA_CY_SYM_HASH_SHA384 result

1.5.11 Define CPA_CY_HKDF_KEY_MAX_LABEL_COUNT

- Defined in file_api_lac_cpa_cy_key.h

Define Documentation

`CPA_CY_HKDF_KEY_MAX_LABEL_COUNT`

space in bytes of largest label for TLS 1.3

1.5.12 Define CPA_CY_HKDF_KEY_MAX_LABEL_SZ

- Defined in file_api_lac_cpa_cy_key.h

Define Documentation

`CPA_CY_HKDF_KEY_MAX_LABEL_SZ`

space in bytes of largest info needed for TLS 1.3, rounded up to multiple of 8

1.5.13 Define CPA_CY_HKDF_KEY_MAX_SECRET_SZ

- Defined in file_api_lac_cpa_cy_key.h

Define Documentation

`CPA_CY_HKDF_KEY_MAX_SECRET_SZ`

1.5.14 Define CPA_CY_HKDF_SUBLABEL_FINISHED

- Defined in file_api_lac_cpa_cy_key.h

Define Documentation

`CPA_CY_HKDF_SUBLABEL_FINISHED`

Bit for creation of key material for 'finished' sublabel

1.5.15 Define CPA_CY_HKDF_SUBLABEL_IV

- Defined in file_api_lac_cpa_cy_key.h

Define Documentation

`CPA_CY_HKDF_SUBLABEL_IV`

Bit for creation of key material for 'iv' sublabel

1.5.16 Define CPA_CY_HKDF_SUBLABEL_KEY

- Defined in file_api_lac_cpa_cy_key.h

Define Documentation

`CPA_CY_HKDF_SUBLABEL_KEY`

TLS Operation Types

These definitions provide bit settings for sublabels for HKDF-ExpandLabel operations.

File: cpa_cy_key.h

Description: Bitwise constants for HKDF sublabels

key sublabel to generate "key" keying material

iv sublabel to generate "iv" keying material

resumption sublabel to generate "resumption" keying material

finished sublabel to generate "finished" keying material Bit for creation of key material for 'key' sub-label

1.5.17 Define CPA_CY_HKDF_SUBLABEL_RESUMPTION

- Defined in file_api_lac_cpa_cy_key.h

Define Documentation

`CPA_CY_HKDF_SUBLABEL_RESUMPTION`

Bit for creation of key material for 'resumption' sublabel

1.5.18 Define CPA_CY_KEY_GEN_SSL_TLS_RANDOM_LEN_IN_BYTES

- Defined in file_api_lac_cpa_cy_key.h

Define Documentation

`CPA_CY_KEY_GEN_SSL_TLS_RANDOM_LEN_IN_BYTES`

SSL or TLS key generation random number length.

Description: 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.

1.5.19 Define CPA_CY_KPT_MAX_AAD_LENGTH

- Defined in file_api_lac_cpa_cy_kpt.h

Define Documentation

`CPA_CY_KPT_MAX_AAD_LENGTH`

Max length of Additional Authenticated Data

See also:

cpaCyKptUnwrapContext

Description: Defines the permitted max aad length in bytes that may be used in private key wrapping/unwrapping.

1.5.20 Define CPA_CY_KPT_MAX_IV_LENGTH

- Defined in file_api_lac_cpa_cy_kpt.h

Define Documentation

`CPA_CY_KPT_MAX_IV_LENGTH`

Max length of initialization vector

See also:

cpaCyKptUnwrapContext

Description: 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.

1.5.21 Define CPA_CY_RSA3K_SIG_SIZE_INBYTES

- Defined in file_api_lac_cpa_cy_kpt.h

Define Documentation

`CPA_CY_RSA3K_SIG_SIZE_INBYTES`

PKCS#1 v2.2 RSA-3K signature output length in bytes.

See also:

CpaCyKptValidationKey

1.5.22 Define CPA_CY_SYM_CCM_SET_AAD

- Defined in file_api_lac_cpa_cy_sym.h

Define Documentation

`CPA_CY_SYM_CCM_SET_AAD`(pOpData, pAad, aadLen)

Setup the additional authentication data for CCM.

Description: This macro sets the additional authentication data in the appropriate location of the*CpaCySymOpData* struct for the authenticated encryption algorithm *CPA_CY_SYM_HASH_AES_CCM*.

1.5.23 Define CPA_CY_SYM_CCM_SET_NONCE

- Defined in file_api_lac_cpa_cy_sym.h

Define Documentation

`CPA_CY_SYM_CCM_SET_NONCE`(pOpData, pNonce, nonceLen)

Setup the nonce for CCM.

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

1.5.24 Define CPA_CY_SYM_CIPHER_CAP_BITMAP_SIZE

- Defined in file_api_lac_cpa_cy_sym.h

Define Documentation

`CPA_CY_SYM_CIPHER_CAP_BITMAP_SIZE`

Size of bitmap needed for cipher "capabilities" type.

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).

Description: 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 *CpaCySymCipherAlgorithm*, so that the value of the enum constant can also be used as the bit position in the bitmap.

1.5.25 Define CPA_CY_SYM_HASH_CAP_BITMAP_SIZE

- Defined in file_api_lac_cpa_cy_sym.h

Define Documentation

`CPA_CY_SYM_HASH_CAP_BITMAP_SIZE`

Size of bitmap needed for hash "capabilities" type.

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).

Description: 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.

1.5.26 Define CPA_CY_SYM_SESUPD_AUTH_KEY

- Defined in file_api_lac_cpa_cy_sym.h

Define Documentation

`CPA_CY_SYM_SESUPD_AUTH_KEY`

1.5.27 Define CPA_CY_SYM_SESUPD_CIPHER_DIR

- Defined in file_api_lac_cpa_cy_sym.h

Define Documentation

`CPA_CY_SYM_SESUPD_CIPHER_DIR`

1.5.28 Define CPA_CY_SYM_SESUPD_CIPHER_KEY

- Defined in file_api_lac_cpa_cy_sym.h

Define Documentation

`CPA_CY_SYM_SESUPD_CIPHER_KEY`

1.5.29 Define CPA_DC_API_VERSION_AT_LEAST

- Defined in file_api_dc_cpa_dc.h

Define Documentation

`CPA_DC_API_VERSION_AT_LEAST`(major, minor)

CPA DC API version at least

File: cpa_dc.h

Description: The minimal supported CPA_DC API version. Allow to check if the API version is equal or above some version to avoid compilation issues with an older API version.

1.5.30 Define CPA_DC_API_VERSION_LESS_THAN

- Defined in file_api_dc_cpa_dc.h

Define Documentation

`CPA_DC_API_VERSION_LESS_THAN`(major, minor)

CPA DC API version less than

File: cpa_dc.h

Description: The maximum supported CPA_DC API version. Allow to check if the API version is below some version to avoid compilation issues with a newer API version.

1.5.31 Define CPA_DC_API_VERSION_NUM_MAJOR

- Defined in file_api_dc_cpa_dc.h

Define Documentation

`CPA_DC_API_VERSION_NUM_MAJOR`

CPA Dc Major Version Number

Description: The CPA_DC API major version number. This number will be incremented when significant churn to the API has occurred. The combination of the major and minor number definitions represent the complete version number for this interface.

1.5.32 Define CPA_DC_API_VERSION_NUM_MINOR

- Defined in file_api_dc_cpa_dc.h

Define Documentation

`CPA_DC_API_VERSION_NUM_MINOR`

CPA DC Minor Version Number

Description: The CPA_DC API minor version number. This number will be incremented when minor changes to the API has occurred. The combination of the major and minor number definitions represent the complete version number for this interface.

1.5.33 Define CPA_DC_BAD_DATA

- Defined in file_api_dc_cpa_dc.h

Define Documentation

`CPA_DC_BAD_DATA`

Service specific return codes

Description: Compression specific return codes Input data in invalid

1.5.34 Define CPA_DC_CHAIN_CAP_BITMAP_SIZE

- Defined in file_api_dc_cpa_dc.h

Define Documentation

`CPA_DC_CHAIN_CAP_BITMAP_SIZE`

Size of bitmap needed for compression chaining capabilities.

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).

Description: Defines the number of bits in the bitmap to represent supported chaining capabilities dcChainCapInfo. Should be set to at least one greater than the largest value in the enumerated type *CpaDcChainOperations*, so that the value of the enum constant can also be used as the bit position in the bitmap.

1.5.35 Define CPA_DEPRECATED

- Defined in file_api_cpa_types.h

Define Documentation

`CPA_DEPRECATED`

Instance State

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

Deprecated:

As of v1.3 of the Crypto API, this enum has been deprecated, replaced by *CpaOperational-State*.

Instance Info Structure

Description: Enumeration of the different instance states that are possible.

Deprecated:

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

EC Point Verification Operation Data.

Description: Structure that contains the information to describe the instance.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

cpaCyEcPointVerify()

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.

1.5.36 Define CPA_DP_BUFLIST

- Defined in file_api_cpa.h

Define Documentation

`CPA_DP_BUFLIST`

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.

1.5.37 Define CPA_INST_ID_SIZE

- Defined in file_api_cpa.h

Define Documentation

`CPA_INST_ID_SIZE`

1.5.38 Define CPA_INST_NAME_SIZE

- Defined in file_api_cpa.h

Define Documentation

`CPA_INST_NAME_SIZE`

Maximum length of the instance name.

1.5.39 Define CPA_INST_PART_NAME_SIZE

- Defined in file_api_cpa.h

Define Documentation

`CPA_INST_PART_NAME_SIZE`

Maximum length of the part name.

1.5.40 Define CPA_INST_SW_VERSION_SIZE

- Defined in file_api_cpa.h

Define Documentation

`CPA_INST_SW_VERSION_SIZE`

Maximum length of the software version string.

1.5.41 Define CPA_INST_VENDOR_NAME_SIZE

- Defined in file_api_cpa.h

Define Documentation

`CPA_INST_VENDOR_NAME_SIZE`

Maximum length of the vendor name.

1.5.42 Define CPA_INSTANCE_HANDLE_SINGLE

- Defined in file_api_cpa.h

Define Documentation

`CPA_INSTANCE_HANDLE_SINGLE`

Default instantiation handle value where there is only a single instance

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

1.5.43 Define CPA_INSTANCE_MAX_ID_SIZE_IN_BYTES

- Defined in file_api_cpa.h

Define Documentation

`CPA_INSTANCE_MAX_ID_SIZE_IN_BYTES`

Maximum instance info id string length in bytes

1.5.44 Define CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES

- Defined in file_api_cpa.h

Define Documentation

`CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES`

Maximum instance info name string length in bytes

1.5.45 Define CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES

- Defined in file_api_cpa.h

Define Documentation

`CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES`

Maximum instance info version string length in bytes

1.5.46 Define CPA_MAX_CORES

- Defined in file_api_cpa.h

Define Documentation

`CPA_MAX_CORES`

Maximum number of cores to support in the coreAffinity bitmap.

1.5.47 Define CPA_STATUS_FAIL

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_FAIL`

Fail status value.

1.5.48 Define CPA_STATUS_FATAL

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_FATAL`

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

1.5.49 Define CPA_STATUS_INVALID_PARAM

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_INVALID_PARAM`

Invalid parameter has been passed in.

1.5.50 Define CPA_STATUS_MAX_STR_LENGTH_IN_BYTES

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_MAX_STR_LENGTH_IN_BYTES`

API status string type definition

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

Description: 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*.

1.5.51 Define CPA_STATUS_RESOURCE

- Defined in file_api_cpa.h

Define Documentation

`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.

1.5.52 Define CPA_STATUS_RESTARTING

- Defined in file_api_cpa.h

Define Documentation

`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.

1.5.53 Define CPA_STATUS_RETRY

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_RETRY`

Retry status value.

1.5.54 Define CPA_STATUS_STR_FAIL

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_STR_FAIL`

Status string for *CPA_STATUS_FAIL*.

1.5.55 Define CPA_STATUS_STR_FATAL

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_STR_FATAL`

Status string for *CPA_STATUS_FATAL*.

1.5.56 Define CPA_STATUS_STR_INVALID_PARAM

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_STR_INVALID_PARAM`

Status string for *CPA_STATUS_INVALID_PARAM*.

1.5.57 Define CPA_STATUS_STR_RESOURCE

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_STR_RESOURCE`

Status string for *CPA_STATUS_RESOURCE*.

1.5.58 Define CPA_STATUS_STR_RETRY

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_STR_RETRY`

Status string for *CPA_STATUS_RETRY*.

1.5.59 Define CPA_STATUS_STR_SUCCESS

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_STR_SUCCESS`

Status string for *CPA_STATUS_SUCCESS*.

1.5.60 Define CPA_STATUS_STR_UNSUPPORTED

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_STR_UNSUPPORTED`

Status string for *CPA_STATUS_UNSUPPORTED*.

1.5.61 Define CPA_STATUS_SUCCESS

- Defined in file_api_cpa.h

Define Documentation

`CPA_STATUS_SUCCESS`

Success status value.

1.5.62 Define CPA_STATUS_UNSUPPORTED

- Defined in file_api_cpa.h

Define Documentation

`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.

1.5.63 Define NULL

- Defined in file_api_cpa_types.h

Define Documentation

`NULL`

NULL definition.

File: cpa_types.h

1.6 Typedefs

1.6.1 Typedef Cpa16S

- Defined in file_api_cpa_types.h

Typedef Documentation

typedef int16_t `Cpa16S`

Signed double-byte base type.

File: cpa_types.h

1.6.2 Typedef Cpa16U

- Defined in file_api_cpa_types.h

Typedef Documentation

typedef uint16_t `Cpa16U`

Unsigned double-byte base type.

File: cpa_types.h

1.6.3 Typedef Cpa32S

- Defined in file_api_cpa_types.h

Typedef Documentation

typedef int32_t `Cpa32S`

Signed quad-byte base type.

File: cpa_types.h

1.6.4 Typedef Cpa32U

- Defined in file_api_cpa_types.h

Typedef Documentation

typedef uint32_t `Cpa32U`

Unsigned quad-byte base type.

File: cpa_types.h

1.6.5 Typedef Cpa64S

- Defined in file_api_cpa_types.h

Typedef Documentation

typedef int64_t `Cpa64S`

Signed double-quad-byte base type.

File: cpa_types.h

1.6.6 Typedef Cpa64U

- Defined in file_api_cpa_types.h

Typedef Documentation

typedef uint64_t `Cpa64U`

Unsigned double-quad-byte base type.

File: cpa_types.h

1.6.7 Typedef Cpa8S

- Defined in file_api_cpa_types.h

Typedef Documentation

typedef int8_t `Cpa8S`

Signed byte base type.

File: cpa_types.h

1.6.8 Typedef Cpa8U

- Defined in file_api_cpa_types.h

Typedef Documentation

typedef uint8_t `Cpa8U`

Unsigned byte base type.

File: cpa_types.h

1.6.9 Typedef CPA_DEPRECATED

- Defined in file_api_cpa.h

Typedef Documentation

typedef enum *_CpaInstanceType* `CPA_DEPRECATED`

Instance Types

Deprecated:

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

Instance State

Description: Enumeration of the different instance types.

Deprecated:

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

Instance Info Structure

Description: Enumeration of the different instance states that are possible.

Deprecated:

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

Description: Structure that contains the information to describe the instance.

1.6.10 Typedef CPA_DEPRECATED

- Defined in file_api_lac_cpa_cy_dh.h

Typedef Documentation

typedef enum *_CpaInstanceType* `CPA_DEPRECATED`

Instance Types

Deprecated:

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

Instance State

Description: Enumeration of the different instance types.

Deprecated:

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

Instance Info Structure

Description: Enumeration of the different instance states that are possible.

Deprecated:

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by CpaInstance-Info2.

Description: Structure that contains the information to describe the instance.

1.6.11 Typedef CPA_DEPRECATED

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef enum *_CpaInstanceType* `CPA_DEPRECATED`

Instance Types

Deprecated:

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

Instance State

Description: Enumeration of the different instance types.

Deprecated:

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

Instance Info Structure

Description: Enumeration of the different instance states that are possible.

Deprecated:

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

Description: Structure that contains the information to describe the instance.

1.6.12 Typedef CPA_DEPRECATED

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef enum *_CpaInstanceType* CPA_DEPRECATED

Instance Types

Deprecated:

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

Instance State

Description: Enumeration of the different instance types.

Deprecated:

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

Instance Info Structure

Description: Enumeration of the different instance states that are possible.

Deprecated:

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

Description: Structure that contains the information to describe the instance.

1.6.13 Typedef CPA_DEPRECATED

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef enum *_CpaInstanceType* CPA_DEPRECATED

Instance Types

Deprecated:

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

Instance State

Description: Enumeration of the different instance types.

Deprecated:

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

Instance Info Structure

Description: Enumeration of the different instance states that are possible.

Deprecated:

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

Description: Structure that contains the information to describe the instance.

1.6.14 Typedef CPA_DEPRECATED

- Defined in file_api_lac_cpa_cy_ln.h

Typedef Documentation

typedef enum *_CpaInstanceType* `CPA_DEPRECATED`

Instance Types

Deprecated:

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

Instance State

Description: Enumeration of the different instance types.

Deprecated:

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

Instance Info Structure

Description: Enumeration of the different instance states that are possible.

Deprecated:

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

Description: Structure that contains the information to describe the instance.

1.6.15 Typedef CPA_DEPRECATED

- Defined in file_api_lac_cpa_cy_prime.h

Typedef Documentation

typedef enum *_CpaInstanceType* `CPA_DEPRECATED`

Instance Types

Deprecated:

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

Instance State

Description: Enumeration of the different instance types.

Deprecated:

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

Instance Info Structure

Description: Enumeration of the different instance states that are possible.

Deprecated:

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

Description: Structure that contains the information to describe the instance.

1.6.16 Typedef CPA_DEPRECATED

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef enum *_CpaInstanceType* `CPA_DEPRECATED`

Instance Types

Deprecated:

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

Instance State

Description: Enumeration of the different instance types.

Deprecated:

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

Instance Info Structure

Description: Enumeration of the different instance states that are possible.

Deprecated:

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

Description: Structure that contains the information to describe the instance.

1.6.17 Typedef CPA_DEPRECATED

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef enum *_CpaInstanceType* `CPA_DEPRECATED`

Instance Types

Deprecated:

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

Instance State

Description: Enumeration of the different instance types.

Deprecated:

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

Instance Info Structure

Description: Enumeration of the different instance states that are possible.

Deprecated:

As of v1.3 of the Crypto API, this structure has been deprecated, replaced by CpaInstance-Info2.

Description: Structure that contains the information to describe the instance.

1.6.18 Typedef CpaAccelerationServiceType

- Defined in file_api_cpa.h

Typedef Documentation

typedef enum *_CpaAccelerationServiceType* `CpaAccelerationServiceType`

Service Type

Description: Enumeration of the different service types.

1.6.19 Typedef CpaBoolean

- Defined in file_api_cpa_types.h

Typedef Documentation

typedef enum *_CpaBoolean* `CpaBoolean`

Boolean type.

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

1.6.20 Typedef CpaBufferList

- Defined in file_api_cpa.h

Typedef Documentation

typedef struct *_CpaBufferList* `CpaBufferList`

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

Description: 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.

1.6.21 Typedef CpaCrcControlData

- Defined in file_api_cpa.h

Typedef Documentation

typedef struct *_CpaCrcControlData* `CpaCrcControlData`

Crc Control data structure for programmable CRC engine.

Description: This structure specifies CRC algorithm parameters which are used to configure a programmable CRC engine. It can be used to specify a CRC algorithm, other than those natively supported by the API.

1.6.22 Typedef CpaCrcData

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaCrcData* `CpaCrcData`

Collection of CRC related data

Description: This structure contains data facilitating CRC calculations. After successful request, this structure will contain all resulting CRCs. Integrity specific CRCs (when enabled/supported) are located in 'CpaIntegrityCrc integrityCrc' field for 32bit values and in 'CpaIntegrityCrc64b integrityCrC64b' field for 64 bit values. Integrity CRCs cannot be accumulated across multiple requests and do not provide seeding capabilities.

Note: this structure must be allocated in physical contiguous memory

---

1.6.23 Typedef CpaCyCapabilitiesInfo

- Defined in file_api_lac_cpa_cy_im.h

Typedef Documentation

typedef struct *_CpaCyCapabilitiesInfo* `CpaCyCapabilitiesInfo`

Cryptographic Capabilities Info

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.

Description: 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.

1.6.24 Typedef CpaCyDhPhase1KeyGenOpData

- Defined in file_api_lac_cpa_cy_dh.h

Typedef Documentation

typedef struct *_CpaCyDhPhase1KeyGenOpData* `CpaCyDhPhase1KeyGenOpData`

Diffie-Hellman Phase 1 Key Generation Data.

Description: 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.

1.6.25 Typedef CpaCyDhPhase2SecretKeyGenOpData

- Defined in file_api_lac_cpa_cy_dh.h

Typedef Documentation

typedef struct *_CpaCyDhPhase2SecretKeyGenOpData* `CpaCyDhPhase2SecretKeyGenOpData`

Diffie-Hellman Phase 2 Secret Key Generation Data.

Description: 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.

1.6.26 Typedef CpaCyDhStats64

- Defined in file_api_lac_cpa_cy_dh.h

Typedef Documentation

typedef struct *_CpaCyDhStats64* `CpaCyDhStats64`

Diffie-Hellman Statistics (64-bit version).

Description: 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.

1.6.27 Typedef CpaCyDsaGenCbFunc

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef void (*`CpaCyDsaGenCbFunc`)(void *pCallbackTag, *CpaStatus* status, void *pOpData, *CpaBoolean* protocolStatus, *CpaFlatBuffer* *pOut)

Definition of a generic callback function invoked for a number of the DSA API functions..

See also:

cpaCyDsaGenPParam() cpaCyDsaGenGParam() cpaCyDsaSignR() cpaCyDsaSignS()

Description: 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

Note: None

Param pCallbackTag [in] User-supplied value to help identify request.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pOpData [in] Opaque pointer to Operation data supplied in request.

Param protocolStatus [in] The result passes/fails the DSA protocol related checks.

Param pOut [in] Output data from the request.

Retval None

Pre Component has been initialized.

Post None

1.6.28 Typedef CpaCyDsaGParamGenOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef struct *_CpaCyDsaGParamGenOpData* `CpaCyDsaGParamGenOpData`

DSA G Parameter Generation Operation Data.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

Description: 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 numbers MUST be stored in big-endian order.

See also:

cpaCyDsaGenGParam()

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.

1.6.29 Typedef CpaCyDsaPParamGenOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef struct *_CpaCyDsaPParamGenOpData* `CpaCyDsaPParamGenOpData`

DSA P Parameter Generation Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. X.pData[0] = MSB.

See also:

cpaCyDsaGenPParam()

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.

1.6.30 Typedef CpaCyDsaRSignOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef struct *_CpaCyDsaRSignOpData* `CpaCyDsaRSignOpData`

DSA R Sign Operation Data.

For optimal performance all data SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

See also:

cpaCyDsaSignR()

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.

1.6.31 Typedef CpaCyDsaRSSignCbFunc

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef void (*`CpaCyDsaRSSignCbFunc`)(void *pCallbackTag, *CpaStatus* status, void *pOpData, *CpaBoolean* protocolStatus, *CpaFlatBuffer* *pR, *CpaFlatBuffer* *pS)

Definition of callback function invoked for cpaCyDsaSignRS requests.

See also:

cpaCyDsaSignRS()

Description: 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

Note: None

Param pCallbackTag [in] User-supplied value to help identify request.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pOpData [in] Operation data pointer supplied in request.

Param protocolStatus [in] The result passes/fails the DSA protocol related checks.

Param pR [in] DSA message signature r.

Param pS [in] DSA message signature s.

Retval None

Pre Component has been initialized.

Post None

1.6.32 Typedef CpaCyDsaRSSignOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef struct *_CpaCyDsaRSSignOpData* `CpaCyDsaRSSignOpData`

DSA R & S Sign Operation Data.

For optimal performance all data SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

See also:

cpaCyDsaSignRS()

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.

1.6.33 Typedef CpaCyDsaSSignOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef struct *_CpaCyDsaSSignOpData* `CpaCyDsaSSignOpData`

DSA S Sign Operation Data.

For optimal performance all data SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. Q.pData[0] = MSB.

See also:

cpaCyDsaSignS()

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.

1.6.34 Typedef CpaCyDsaStats64

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef struct *_CpaCyDsaStats64* `CpaCyDsaStats64`

Cryptographic DSA Statistics (64-bit version).

Description: 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.

1.6.35 Typedef CpaCyDsaVerifyCbFunc

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef void (*`CpaCyDsaVerifyCbFunc`)(void *pCallbackTag, *CpaStatus* status, void *pOpData, *CpaBoolean* verifyStatus)

Definition of callback function invoked for cpaCyDsaVerify requests.

See also:

cpaCyDsaVerify()

Description: 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

Note: None

Param pCallbackTag [in] User-supplied value to help identify request.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pOpData [in] Operation data pointer supplied in request.

Param verifyStatus [in] The verification passed or failed.

Retval None

Pre Component has been initialized.

Post None

1.6.36 Typedef CpaCyDsaVerifyOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef struct *_CpaCyDsaVerifyOpData* `CpaCyDsaVerifyOpData`

DSA Verify Operation Data.

For optimal performance all data SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

See also:

cpaCyDsaVerify()

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.

1.6.37 Typedef CpaCyDsaYParamGenOpData

- Defined in file_api_lac_cpa_cy_dsa.h

Typedef Documentation

typedef struct *_CpaCyDsaYParamGenOpData* `CpaCyDsaYParamGenOpData`

DSA Y Parameter Generation Operation Data.

For optimal performance all data SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. P.pData[0] = MSB.

See also:

cpaCyDsaGenYParam()

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.

1.6.38 Typedef CpaCyEcCurve

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef struct *_CpaCyEcCurve* `CpaCyEcCurve`

Unified curve parameters.

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.

Description: This structure provides a single data type that can describe a number of different curve types. The intention is to add further curve types in the future, thus the union field will allow for that expansion.

For optimal performance all data buffers SHOULD be 8-byte aligned.

See also:

CpaCyEcCurveParameters cpaCyEcGenericPointMultiply cpaCyEcGenericPointVerify

---

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

1.6.39 Typedef CpaCyEcCurveParameters

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef union *_CpaCyEcCurveParameters* `CpaCyEcCurveParameters`

Union characterised by a specific curve.

See also:

CpaCyEcCurveParametersWeierstrass

Description: This union allows for the characterisation of different curve types encapsulted in one data type. The intention is that new curve types will be added in the future.

Note:

1.6.40 Typedef CpaCyEcCurveParametersWeierstrass

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef struct *_CpaCyEcCurveParametersWeierstrass* `CpaCyEcCurveParametersWeierstrass`

Curve parameters for a Weierstrass type curve.

For optimal performance all data buffers SHOULD be 8-byte aligned. The legend used in this structure is borrowed from RFC7748

Description: This structure contains curve parameters for Weierstrass type curve: y^2 = x^3 + ax + b 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 function.

See also:

CpaCyEcCurveParameters CpaCyEcFieldType

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

1.6.41 Typedef CpaCyEcCurveType

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef enum *_CpaCyEcCurveType* `CpaCyEcCurveType`

Enumeration listing curve types to use with generic multiplication and verification routines.

See also:

cpaCyEcGenericPointMultiply() cpaCyEcGenericPointVerify()

Description: This structure contains a list of different elliptic curve types. EC Point multiplication and other operations depend on the type of the curve.

1.6.42 Typedef CpaCyEcdsaGenSignCbFunc

- Defined in file_api_lac_cpa_cy_ecdsa.h

Typedef Documentation

typedef void (*`CpaCyEcdsaGenSignCbFunc`)(void *pCallbackTag, *CpaStatus* status, void *pOpData, *CpaBoolean* multiplyStatus, *CpaFlatBuffer* *pOut)

Definition of a generic callback function invoked for a number of the ECDSA Sign API functions.

See also:

cpaCyEcdsaSignR() cpaCyEcdsaSignS()

Description: 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

Note: None

Param pCallbackTag [in] User-supplied value to help identify request.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pOpData [in] Opaque pointer to Operation data supplied in request.

Param multiplyStatus [in] Status of the point multiplication.

Param pOut [in] Output data from the request.

Retval None

Pre Component has been initialized.

Post None

1.6.43 Typedef CpaCyEcdsaSignROpData

- Defined in file_api_lac_cpa_cy_ecdsa.h

Typedef Documentation

typedef struct *_CpaCyEcdsaSignROpData* `CpaCyEcdsaSignROpData`

ECDSA Sign R Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

cpaCyEcdsaSignR()

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.

1.6.44 Typedef CpaCyEcdsaSignRSCbFunc

- Defined in file_api_lac_cpa_cy_ecdsa.h

Typedef Documentation

typedef void (*`CpaCyEcdsaSignRSCbFunc`)(void *pCallbackTag, *CpaStatus* status, void *pOpData, *CpaBoolean* multiplyStatus, *CpaFlatBuffer* *pR, *CpaFlatBuffer* *pS)

Definition of callback function invoked for cpaCyEcdsaSignRS requests.

See also:

cpaCyEcdsaSignRS()

Description: 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

Note: None

Param pCallbackTag [in] User-supplied value to help identify request.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pOpData [in] Operation data pointer supplied in request.

Param multiplyStatus [in] Status of the point multiplication.

Param pR [in] Ecdsa message signature r.

Param pS [in] Ecdsa message signature s.

Retval None

Pre Component has been initialized.

Post None

1.6.45 Typedef CpaCyEcdsaSignRSOpData

- Defined in file_api_lac_cpa_cy_ecdsa.h

Typedef Documentation

typedef struct *_CpaCyEcdsaSignRSOpData* `CpaCyEcdsaSignRSOpData`

ECDSA Sign R & S Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

cpaCyEcdsaSignRS()

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.

1.6.46 Typedef CpaCyEcdsaSignSOpData

- Defined in file_api_lac_cpa_cy_ecdsa.h

Typedef Documentation

typedef struct *_CpaCyEcdsaSignSOpData* `CpaCyEcdsaSignSOpData`

ECDSA Sign S Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

cpaCyEcdsaSignS()

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.

1.6.47 Typedef CpaCyEcdsaStats64

- Defined in file_api_lac_cpa_cy_ecdsa.h

Typedef Documentation

typedef struct *_CpaCyEcdsaStats64* `CpaCyEcdsaStats64`

Cryptographic ECDSA Statistics.

Description: This structure contains statistics on the Cryptographic ECDSA operations. Statistics are set to zero when the component is initialized, and are collected per instance.

1.6.48 Typedef CpaCyEcdsaVerifyCbFunc

- Defined in file_api_lac_cpa_cy_ecdsa.h

Typedef Documentation

typedef void (*`CpaCyEcdsaVerifyCbFunc`)(void *pCallbackTag, *CpaStatus* status, void *pOpData, *CpaBoolean* verifyStatus)

Definition of callback function invoked for cpaCyEcdsaVerify requests.

See also:

cpaCyEcdsaVerify()

Description: 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

Note: None

Param pCallbackTag [in] User-supplied value to help identify request.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pOpData [in] Operation data pointer supplied in request.

Param verifyStatus [in] The verification status.

Retval None

Pre Component has been initialized.

Post None

1.6.49 Typedef CpaCyEcdsaVerifyOpData

- Defined in file_api_lac_cpa_cy_ecdsa.h

Typedef Documentation

typedef struct *_CpaCyEcdsaVerifyOpData* `CpaCyEcdsaVerifyOpData`

ECDSA Verify Operation Data, for Public Key.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: 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.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

CpaCyEcdsaVerify()

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.

1.6.50 Typedef CpaCyEcFieldType

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef enum *_CpaCyEcFieldType* `CpaCyEcFieldType`

Field types for Elliptic Curve

Description: 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.

1.6.51 Typedef CpaCyEcGenericPointMultiplyOpData

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef struct *_CpaCyEcGenericPointMultiplyOpData* `CpaCyEcGenericPointMultiplyOpData`

Generic EC Point Multiplication Operation Data.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: This structure contains a generic EC point and a multiplier for use with cpaCyEcGenericPointMultiply. This is common for representing all EC points, irrespective of curve type: Weierstrass, Montgomery and Twisted Edwards (at this time only Weierstrass are supported). The same point + multiplier format can be used when performing generator multiplication, in which case the xP, yP supplied in this structure will be ignored by QAT API library & a generator point will be inserted in their place.

All values in this structure are required to be in Most Significant Byte first order, e.g. a.pData[0] = MSB.

See also:

cpaCyEcGenericPointMultiply()

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

1.6.52 Typedef CpaCyEcGenericPointVerifyOpData

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef struct *_CpaCyEcGenericPointVerifyOpData* `CpaCyEcGenericPointVerifyOpData`

Generic EC Point Verify Operation Data.

This structure contains a generic EC point, irrespective of curve type. It is used to verify when the <x,y> pair specified in the structure lies on the curve indicated in the cpaCyEcGenericPointVerify API.

Description: This structure contains the operation data for the cpaCyEcGenericPointVerify function. This is common for representing all EC points, irrespective of curve type: Weierstrass, Montgomery and Twisted Edwards (at this time only Weierstrass are supported).

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.

See also:

cpaCyEcGenericPointVerify()

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

1.6.53 Typedef CpaCyEcMontEdwdsCurveType

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef enum *_CpaCyEcMontEdwdsCurveType* `CpaCyEcMontEdwdsCurveType`

Curve types for Elliptic Curves defined in RFC#7748

Description: As defined by RFC 7748, there are four elliptic curves in this group. The Montgomery curves are denoted curve25519 and curve448, and the birationally equivalent Twisted Edwards curves are denoted edwards25519 and edwards448

1.6.54 Typedef CpaCyEcMontEdwdsPointMultiplyOpData

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef struct *_CpaCyEcMontEdwdsPointMultiplyOpData* `CpaCyEcMontEdwdsPointMultiplyOpData`

EC Point Multiplication Operation Data for Edwards or Montgomery curves as specificied in RFC#7748.

For optimal performance all data buffers SHOULD be 8-byte aligned.

Description: This structure contains the operation data for the cpaCyEcMontEdwdsPointMultiply 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. a.pData[0] = MSB.

All buffers in this structure need to be:

- 32 bytes in size for 25519 curves
- 64 bytes in size for 448 curves

See also:

cpaCyEcMontEdwdsPointMultiply()

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

1.6.55 Typedef CpaCyEcPointMultiplyCbFunc

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef void (*`CpaCyEcPointMultiplyCbFunc`)(void *pCallbackTag, *CpaStatus* status, void *pOpData, *CpaBoolean* multiplyStatus, *CpaFlatBuffer* *pXk, *CpaFlatBuffer* *pYk)

Definition of callback function invoked for cpaCyEcPointMultiply requests.

See also:

cpaCyEcGenericPointMultiply()

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

Note: None

Param pCallbackTag [in] User-supplied value to help identify request.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pOpData [in] Opaque pointer to Operation data supplied in request.

Param multiplyStatus [in] Status of the point multiplication.

Param pXk [in] x coordinate of resultant EC point.

Param pYk [in] y coordinate of resultant EC point.

Retval None

Pre Component has been initialized.

Post None

1.6.56 Typedef CpaCyEcPointVerifyCbFunc

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef void (*`CpaCyEcPointVerifyCbFunc`)(void *pCallbackTag, *CpaStatus* status, void *pOpData, *CpaBoolean* verifyStatus)

Definition of callback function invoked for cpaCyEcGenericPointVerify requests.

See also:

cpaCyEcGenericPointVerify()

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

Note: None

Param pCallbackTag [in] User-supplied value to help identify request.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pOpData [in] Operation data pointer supplied in request.

Param verifyStatus [in] 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 None

Pre Component has been initialized.

Post None

1.6.57 Typedef CpaCyEcStats64

- Defined in file_api_lac_cpa_cy_ec.h

Typedef Documentation

typedef struct *_CpaCyEcStats64* `CpaCyEcStats64`

Cryptographic EC Statistics.

Description: This structure contains statistics on the Cryptographic EC operations. Statistics are set to zero when the component is initialized, and are collected per instance.

1.6.58 Typedef CpaCyGenericCbFunc

- Defined in file_api_lac_cpa_cy_common.h

Typedef Documentation

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

Definition of the crypto generic callback function

See also:

cpaCyKeyGenSsl()

Description: 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

Note: None

Param pCallbackTag [in] Opaque value provided by user while making individual function call.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pOpData [in] Opaque Pointer to the operation data that was submitted in the request

Retval None

Pre Component has been initialized.

Post None

1.6.59 Typedef CpaCyGenFlatBufCbFunc

- Defined in file_api_lac_cpa_cy_common.h

Typedef Documentation

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

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

See also:

None

Description: 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

Note: None

Param pCallbackTag [in] Opaque value provided by user while making individual function call.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pOpData [in] Opaque Pointer to the operation data that was submitted in the request

Param pOut [in] Pointer to the output buffer provided in the request invoking this callback.

Retval None

Pre Component has been initialized.

Post None

1.6.60 Typedef CpaCyInstanceNotificationCbFunc

- Defined in file_api_lac_cpa_cy_common.h

Typedef Documentation

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

Callback function for instance notification support.

See also:

cpaCyInstanceSetNotificationCb(),

Description: 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

Note: None

Param instanceHandle [in] Instance handle.

Param pCallbackTag [in] Opaque value provided by user while making individual function calls.

Param instanceEvent [in] The event that will trigger this function to get invoked.

Retval None

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

Post None

1.6.61 Typedef CpaCyKeyGenHKDFExpandLabel

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef struct *_CpaCyKeyGenHKDFExpandLabel* `CpaCyKeyGenHKDFExpandLabel`

Maximum number of labels in op structure

TLS data for key generation functions

File: cpa_cy_key.h

Description: This structure contains data for describing label for the HKDF Extract Label function

Extract Label Function

labelLen = length of the label field

contextLen = length of the context field

sublabelFlag = Mask of sub labels required for this label.

label = label as defined in RFC8446

context = context as defined in RFC8446

1.6.62 Typedef CpaCyKeyGenHKDFOpData

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef struct *_CpaCyKeyGenHKDFOpData* `CpaCyKeyGenHKDFOpData`

TLS data for key generation functions

Description:

This structure contains data for all HKDF operations:

HKDF Extract

HKDF Expand

HKDF Expand Label

HKDF Extract and Expand

HKDF Extract and Expand Label

HKDF Map Structure Elements

secret - IKM value for extract operations or PRK for expand or expand operations.

seed - contains the salt for extract operations

info - contains the info data for extract operations

labels - See notes above

1.6.63 Typedef CpaCyKeyGenMgfOpData

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef struct *_CpaCyKeyGenMgfOpData* `CpaCyKeyGenMgfOpData`

Key Generation Mask Generation Function (MGF) Data

See also:

cpaCyKeyGenMgf

Description: 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*.

1.6.64 Typedef CpaCyKeyGenMgfOpDataExt

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef struct *_CpaCyKeyGenMgfOpDataExt* `CpaCyKeyGenMgfOpDataExt`

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

See also:

cpaCyKeyGenMgfExt

Description: 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.

1.6.65 Typedef CpaCyKeyGenSslOpData

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef struct *_CpaCyKeyGenSslOpData* `CpaCyKeyGenSslOpData`

SSL data for key generation functions

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

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

Note: In each of the above descriptions, + indicates concatenation.

Note: 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.

1.6.66 Typedef CpaCyKeyGenStats64

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef struct *_CpaCyKeyGenStats64* `CpaCyKeyGenStats64`

Key Generation Statistics (64-bit version).

Description: 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.

1.6.67 Typedef CpaCyKeyGenTlsOpData

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef struct *_CpaCyKeyGenTlsOpData* `CpaCyKeyGenTlsOpData`

TLS data for key generation functions

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

Description: 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

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.

Note: In each of the above descriptions, + indicates concatenation.

Note: 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.

1.6.68 Typedef CpaCyKeyHKDFCipherSuite

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef enum *_CpaCyKeyHKDFCipherSuite* `CpaCyKeyHKDFCipherSuite`

TLS Operation Types

The function *cpaCyKeyGenTls3* accelerates the TLS HKDF, which is defined as part of RFC5869 (HKDF) and RFC8446 (TLS v1.3).

File: cpa_cy_key.h

Description: Enumeration of the different cipher suites that may be used in a TLS v1.3 operation. This value is used to infer the sizes of the key and iv sublabel.

This enumerated type defines the supported cipher suites in the TLS operation that require HKDF key operations.

1.6.69 Typedef CpaCyKeyHKDFOp

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef enum *_CpaCyKeyHKDFOp* `CpaCyKeyHKDFOp`

TLS Operation Types

The function *cpaCyKeyGenTls3* accelerates the TLS HKDF, which is defined as part of RFC5869 (HKDF) and RFC8446 (TLS v1.3).

File: cpa_cy_key.h

Description: Enumeration of the different TLS operations that can be specified in the CpaCyKey-GenHKDFOpData.

This enumerated type defines the support HKDF operations for extraction and expansion of keying material.

1.6.70 Typedef CpaCyKeySslOp

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef enum *_CpaCyKeySslOp* `CpaCyKeySslOp`

SSL Operation Types

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

1.6.71 Typedef CpaCyKeyTlsOp

- Defined in file_api_lac_cpa_cy_key.h

Typedef Documentation

typedef enum *_CpaCyKeyTlsOp* `CpaCyKeyTlsOp`

TLS Operation Types

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 each of 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.

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

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.

1.6.72 Typedef CpaCyKptEcdsaSignRSOpData

- Defined in file_api_lac_cpa_cy_kpt.h

Typedef Documentation

typedef struct *CpaCyKptEcdsaSignRSOpData_t* `CpaCyKptEcdsaSignRSOpData`

KPT ECDSA Sign R & S Operation Data.

File: cpa_cy_kpt.h

Key = SWK AAD = DER(OID) IV = nonce Input = (d) Encrypt (SWK, AAD, IV, (d)) Output (AuthTag, EncryptedECKey)

Description: 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. This key structure is encrypted when passed into cpaCyKptEcdsaSignRS Encrypt - AES-256-GCM (Key, AAD, Input) "||" - denotes concatenation

privatekey == EncryptedECKey || AuthTag

OID's that shall be supported by KPT implementation: Curve OID DER(OID) secp256r1 1.2.840.10045.3.1.7 06 08 2A 86 48 CE 3D 03 01 07 secp384r1 1.3.132.0.34 06 05 2B 81 04 00 22 secp521r1 1.3.132.0.35 06 05 2B 81 04 00 23

Expected private key (d) sizes: secp256r1 256 bits secp384r1 384 bits secp521r1 576 bits (rounded up to a multiple of 64-bit quadword)

AuthTag is 128 bits (16 bytes)

For optimal performance all data buffers SHOULD be 8-byte aligned.

See also:

cpaCyEcdsaSignRS()

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.

1.6.73 Typedef CpaCyKptHandle

- Defined in file_api_lac_cpa_cy_kpt.h

Typedef Documentation

typedef *Cpa64U* `CpaCyKptHandle`

KPT wrapping key handle

Description: 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.

1.6.74 Typedef CpaCyKptKeyManagementStatus

- Defined in file_api_lac_cpa_cy_kpt.h

Typedef Documentation

typedef enum *CpaCyKptKeyManagementStatus_t* `CpaCyKptKeyManagementStatus`

Return Status

Description: This enumeration lists all the possible return status after completing KPT APIs.

1.6.75 Typedef CpaCyKptLoadKey

- Defined in file_api_lac_cpa_cy_kpt.h

Typedef Documentation

typedef struct *CpaCyKptLoadKey_t* `CpaCyKptLoadKey`

KPT Loading key format specification.

Description: This structure defines the format of the symmetric wrapping key to be loaded into KPT. Application sets these parameters through the cpaCyKptLoadKey calls.

1.6.76 Typedef CpaCyKptRsaDecryptOpData

- Defined in file_api_lac_cpa_cy_kpt.h

Typedef Documentation

typedef struct *CpaCyKptRsaDecryptOpData_t* `CpaCyKptRsaDecryptOpData`

KPT RSA Decryption Primitive Operation Data

File: cpa_cy_kpt.h

Description: This structure lists the different items that are required in the cpaCyKptRsaDecrypt 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 CpaCyGenFlatBufCbFunc callback function.

Note: If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyKptRsaDecrypt 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.

1.6.77 Typedef CpaCyKptRsaPrivateKey

- Defined in file_api_lac_cpa_cy_kpt.h

Typedef Documentation

typedef struct *CpaCyKptRsaPrivateKey_t* `CpaCyKptRsaPrivateKey`

RSA Private Key Structure.

File: cpa_cy_kpt.h

Description: 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.

1.6.78 Typedef CpaCyKptRsaPrivateKeyRep1

- Defined in file_api_lac_cpa_cy_kpt.h

Typedef Documentation

typedef struct *CpaCyKptRsaPrivateKeyRep1_t* `CpaCyKptRsaPrivateKeyRep1`

RSA Private Key Structure For Representation 1.

privateKey = (EncryptedRSAKey || AuthTag)

File: cpa_cy_kpt.h

Description: 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). The representation is encrypted as follows: Encrypt - AES-256-GCM (Key, AAD, Input) "||" - denotes concatenation Key = SWK AAD = DER(OID) IV = nonce Input = (D || N) Encrypt (SWK, AAD, IV, (D || N)) Output (AuthTag, (D || N)') EncryptedRSAKey = (D || N)'

OID's that shall be supported by KPT implementation: OID DER(OID) 1.2.840.113549.1.1 06 08 2A 86 48 86 F7 0D 01 01

Permitted lengths for N and D are:

- 512 bits (64 bytes),
- 1024 bits (128 bytes),
- 1536 bits (192 bytes),
- 2048 bits (256 bytes),
- 3072 bits (384 bytes),
- 4096 bits (512 bytes), or
- 8192 bits (1024 bytes).

AuthTag is 128 bits (16 bytes)

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. It is critical a unique nonce is used for each SWK encrypt operation.

1.6.79 Typedef CpaCyKptRsaPrivateKeyRep2

- Defined in file_api_lac_cpa_cy_kpt.h

Typedef Documentation

typedef struct *CpaCyKptRsaPrivateKeyRep2_t* `CpaCyKptRsaPrivateKeyRep2`

KPT RSA Private Key Structure For Representation 2.

privateKey = EncryptedRSAKey || AuthTag

File: cpa_cy_kpt.h

Description: 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. For KPT the parameters are Encrypted with the assoicated SWK as follows: Encrypt - AES-256-GCM (Key, AAD, Input) "||" - denotes concatenation Key = SWK IV = nonce AAD = DER(OID) Input = (P || Q || dP || dQ || Qinv || publicExponentE) Expanded Description: Encrypt (SWK, AAD, IV, (P || Q || dP || dQ || Qinv || publicExponentE)) EncryptedRSAKey = (P || Q || dP || dQ || Qinv || publicExponentE)' Output (AuthTag, EncryptedRSAKey)

OID's that shall be supported by KPT implementation: OID DER(OID) 1.2.840.113549.1.1 06 08 2A 86 48 86 F7 0D 01 01

All of the encrypted parameters will be of equal size. The length of each will be equal to keySize in bytes/2. For example for a key size of 256 Bytes (2048 bits), the length of P, Q, dP, dQ, and Qinv are all 128 Bytes, plus the publicExponentE of 256 Bytes, giving a total size for EncryptedRSAKey of 896 Bytes.

AuthTag is 128 bits (16 bytes)

Permitted Key Sizes are:

- 512 bits (64 bytes),
- 1024 bits (128 bytes),
- 1536 bits (192 bytes),
- 2048 bits (256 bytes),
- 3072 bits (384 bytes),
- 4096 bits (512 bytes), or
- 8192 bits (1024 bytes).

Note: It is critical a unique nonce is used for each SWK encrypt operation.

1.6.80 Typedef CpaCyKptUnwrapContext

- Defined in file_api_lac_cpa_cy_kpt.h

Typedef Documentation

typedef struct *CpaCyKptUnwrapContext_t* `CpaCyKptUnwrapContext`

Structure of KPT unwrapping context.

File: cpa_cy_kpt.h

Description: This structure is a parameter of KPT crypto APIs, it contains data relating to KPT WPK unwrapping, the application needs to fill in this information.

1.6.81 Typedef CpaCyKptValidationKey

- Defined in file_api_lac_cpa_cy_kpt.h

Typedef Documentation

typedef struct *CpaCyKptValidationKey_t* `CpaCyKptValidationKey`

KPT device credentials key certificate

See also:

cpaCyKptQueryDeviceCredentials

Description: This structure defines the key format for use with KPT.

1.6.82 Typedef CpaCyKptWrappingKeyType

- Defined in file_api_lac_cpa_cy_kpt.h

Typedef Documentation

typedef enum *CpaCyKptWrappingKeyType_t* `CpaCyKptWrappingKeyType`

Cipher algorithms used to generate a wrapped private key (WPK) from the clear private key.

Description: This enumeration lists supported cipher algorithms and modes.

1.6.83 Typedef CpaCyLnModExpOpData

- Defined in file_api_lac_cpa_cy_ln.h

Typedef Documentation

typedef struct *_CpaCyLnModExpOpData* `CpaCyLnModExpOpData`

Modular Exponentiation Function Operation Data.

The values of the base, the exponent and the modulus MUST all be less than 2^8192, and the modulus must not be equal to zero.

Description: 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.

1.6.84 Typedef CpaCyLnModInvOpData

- Defined in file_api_lac_cpa_cy_ln.h

Typedef Documentation

typedef struct *_CpaCyLnModInvOpData* `CpaCyLnModInvOpData`

Modular Inversion Function Operation Data.

Note that the values of A and B MUST NOT both be even numbers, and both MUST be less than 2^8192.

Description: 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.

1.6.85 Typedef CpaCyLnStats64

- Defined in file_api_lac_cpa_cy_ln.h

Typedef Documentation

typedef struct *_CpaCyLnStats64* `CpaCyLnStats64`

Look Aside Cryptographic large number Statistics.

Description: 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.

1.6.86 Typedef CpaCyPrimeStats64

- Defined in file_api_lac_cpa_cy_prime.h

Typedef Documentation

typedef struct *_CpaCyPrimeStats64* `CpaCyPrimeStats64`

Prime Number Test Statistics (64-bit version).

Description: 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.

1.6.87 Typedef CpaCyPrimeTestCbFunc

- Defined in file_api_lac_cpa_cy_prime.h

Typedef Documentation

typedef void (*`CpaCyPrimeTestCbFunc`)(void *pCallbackTag, *CpaStatus* status, void *pOpData, *CpaBoolean* testPassed)

Definition of callback function invoked for cpaCyPrimeTest requests.

See also:

cpaCyPrimeTest()

Description: 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

Note: None

Param pCallbackTag [in] User-supplied value to help identify request.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pOpData [in] Opaque pointer to the Operation data pointer supplied in request.

Param testPassed [in] A value of CPA_TRUE means the prime candidate is probably prime.

Retval None

Pre Component has been initialized.

Post None

1.6.88 Typedef CpaCyPrimeTestOpData

- Defined in file_api_lac_cpa_cy_prime.h

Typedef Documentation

typedef struct *_CpaCyPrimeTestOpData* `CpaCyPrimeTestOpData`

Prime Test Operation Data.

All values in this structure are required to be in Most Significant Byte first order, e.g. primeCandidate.pData[0] = MSB.

Description: 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 numbers MUST be stored in big-endian order.

See also:

cpaCyPrimeTest()

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.

1.6.89 Typedef CpaCyPriority

- Defined in file_api_lac_cpa_cy_common.h

Typedef Documentation

typedef enum *_CpaCyPriority* `CpaCyPriority`

Request priority

File: cpa_cy_common.h

Description: 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.

1.6.90 Typedef CpaCyRsaDecryptOpData

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef struct *_CpaCyRsaDecryptOpData* `CpaCyRsaDecryptOpData`

RSA Decryption Primitive Operation Data

Description: 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.

1.6.91 Typedef CpaCyRsaEncryptOpData

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef struct *_CpaCyRsaEncryptOpData* `CpaCyRsaEncryptOpData`

RSA Encryption Primitive Operation Data

Description: 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.

1.6.92 Typedef CpaCyRsaKeyGenCbFunc

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef void (*`CpaCyRsaKeyGenCbFunc`)(void *pCallbackTag, *CpaStatus* status, void *pKeyGenOpData, *CpaCyRsaPrivateKey* *pPrivateKey, *CpaCyRsaPublicKey* *pPublicKey)

Definition of the RSA key generation callback function.

See also:

CpaCyRsaPrivateKey, *CpaCyRsaPublicKey*, *cpaCyRsaGenKey()*

Description: 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

Note: None

Param pCallbackTag [in] Opaque value provided by user while making individual function calls.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param pKeyGenOpData [in] Structure with output params for callback.

Param pPrivateKey [in] Structure which contains pointers to the memory into which the generated private key will be written.

Param pPublicKey [in] 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.

Retval None

Pre Component has been initialized.

Post None

1.6.93 Typedef CpaCyRsaKeyGenOpData

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef struct *_CpaCyRsaKeyGenOpData* `CpaCyRsaKeyGenOpData`

RSA Key Generation Data.

The following limitations on the permutations of the supported bit lengths of p, q and n (written as {p, q, n}) apply:

Description: 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.

- {256, 256, 512} or
- {512, 512, 1024} or
- {768, 768, 1536} or
- {1024, 1024, 2048} or
- {1536, 1536, 3072} or
- {2048, 2048, 4096}.

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.

1.6.94 Typedef CpaCyRsaPrivateKey

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef struct *_CpaCyRsaPrivateKey* `CpaCyRsaPrivateKey`

RSA Private Key Structure.

Description: 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.

1.6.95 Typedef CpaCyRsaPrivateKeyRep1

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef struct *_CpaCyRsaPrivateKeyRep1* `CpaCyRsaPrivateKeyRep1`

RSA Private Key Structure For Representation 1.

Description: 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.

1.6.96 Typedef CpaCyRsaPrivateKeyRep2

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef struct *_CpaCyRsaPrivateKeyRep2* `CpaCyRsaPrivateKeyRep2`

RSA Private Key Structure For Representation 2.

Description: 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.

1.6.97 Typedef CpaCyRsaPrivateKeyRepType

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef enum *_CpaCyRsaPrivateKeyRepType* `CpaCyRsaPrivateKeyRepType`

RSA private key representation type.

Description: This enumeration lists which PKCS V2.1 representation of the private key is being used.

1.6.98 Typedef CpaCyRsaPublicKey

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef struct *_CpaCyRsaPublicKey* `CpaCyRsaPublicKey`

RSA Public Key Structure.

Description: 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.

1.6.99 Typedef CpaCyRsaStats64

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef struct *_CpaCyRsaStats64* `CpaCyRsaStats64`

RSA Statistics (64-bit version).

Description: 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.

1.6.100 Typedef CpaCyRsaVersion

- Defined in file_api_lac_cpa_cy_rsa.h

Typedef Documentation

typedef enum *_CpaCyRsaVersion* `CpaCyRsaVersion`

RSA Version.

Description: This enumeration lists the version identifier for the PKCS #1 V2.1 standard.

Note: Multi-prime (more than two primes) is not supported.

1.6.101 Typedef CpaCySymAlgChainOrder

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef enum *_CpaCySymAlgChainOrder* `CpaCySymAlgChainOrder`

Algorithm Chaining Operation Ordering

Description: This enum defines the ordering of operations for algorithm chaining.

1.6.102 Typedef CpaCySymCapabilitiesInfo

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymCapabilitiesInfo* `CpaCySymCapabilitiesInfo`

Symmetric Capabilities Info

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

Description: 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.

```
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.
```

1.6.103 Typedef CpaCySymCbFunc

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

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

Definition of callback function

See also:

cpaCySymInitSession(), *cpaCySymRemoveSession()*

Description: 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

Note: None

Param pCallbackTag **[in]** Opaque value provided by user while making individual function call.

Param status **[in]** Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param operationType **[in]** Identifies the operation type that was requested in the cpaCySymPerformOp function.

Param pOpData **[in]** Pointer to structure with input parameters.

Param pDstBuffer **[in]** 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.

Param verifyResult **[in]** 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.

Retval None

Pre Component has been initialized.

Post None

1.6.104 Typedef CpaCySymCipherAlgorithm

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef enum *_CpaCySymCipherAlgorithm* `CpaCySymCipherAlgorithm`

Cipher algorithms.

Description: This enumeration lists supported cipher algorithms and modes.

1.6.105 Typedef CpaCySymCipherDirection

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef enum *_CpaCySymCipherDirection* `CpaCySymCipherDirection`

Symmetric Cipher Direction

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

1.6.106 Typedef CpaCySymCipherSetupData

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymCipherSetupData* `CpaCySymCipherSetupData`

Symmetric Cipher Setup Data.

Description: This structure contains data relating to Cipher (Encryption and Decryption) to setup a session.

1.6.107 Typedef CpaCySymDeriveOpData

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymDeriveOpData* `CpaCySymDeriveOpData`

Symmetric Cipher Op Data for key derivation

File: cpa_cy_sym.h

Description: This structure contains the cipher key or the data to derive the cipher key in addition to other cipher related data.

1.6.108 Typedef CpaCySymDpCbFunc

- Defined in file_api_lac_cpa_cy_sym_dp.h

Typedef Documentation

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

Definition of callback function for cryptographic data plane API.

See also:

cpaCySymDpRegCbFunc

Description: 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

Note: None

Param pOpData [in] Pointer to the CpaCySymDpOpData object which was supplied as part of the original request.

Param status [in] Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Param verifyResult [in] 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.

Return None

Pre Component has been initialized. Callback has been registered with *cpaCySymDpRegCbFunc*.

Post None

1.6.109 Typedef CpaCySymDpOpData

- Defined in file_api_lac_cpa_cy_sym_dp.h

Typedef Documentation

typedef struct *_CpaCySymDpOpData* `CpaCySymDpOpData`

Operation Data for cryptographic data plane API.

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

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

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

See also:

cpaCySymDpEnqueueOp, cpaCySymDpEnqueueOpBatch

1.6.110 Typedef CpaCySymDpSessionCtx

- Defined in file_api_lac_cpa_cy_sym_dp.h

Typedef Documentation

typedef void *`CpaCySymDpSessionCtx`

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

Description: 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.

1.6.111 Typedef CpaCySymHashAlgorithm

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef enum *_CpaCySymHashAlgorithm* `CpaCySymHashAlgorithm`

Hash algorithms.

Description: This enumeration lists supported hash algorithms.

1.6.112 Typedef CpaCySymHashAuthModeSetupData

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymHashAuthModeSetupData* `CpaCySymHashAuthModeSetupData`

Hash Auth Mode Setup Data.

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

1.6.113 Typedef CpaCySymHashMode

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef enum *_CpaCySymHashMode* `CpaCySymHashMode`

Symmetric Hash mode

Description: This enum indicates the Hash Mode.

1.6.114 Typedef CpaCySymHashNestedModeSetupData

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymHashNestedModeSetupData* `CpaCySymHashNestedModeSetupData`

Hash Mode Nested Setup Data.

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

1.6.115 Typedef CpaCySymHashSetupData

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymHashSetupData* `CpaCySymHashSetupData`

Hash Setup Data.

Description: 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.

1.6.116 Typedef CpaCySymOp

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef enum *_CpaCySymOp* `CpaCySymOp`

Types of operations supported by the cpaCySymPerformOp function.

See also:

cpaCySymPerformOp

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

1.6.117 Typedef CpaCySymOpData

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymOpData* `CpaCySymOpData`

Cryptographic Component Operation Data.

See also:

CpaCySymPacketType

Description: 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.

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.

1.6.118 Typedef CpaCySymOpData2

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymOpData2* `CpaCySymOpData2`

Cryptographic Component Operation Data with additional arguments

If the deriveCtxData structure contains non-NULL entries for the context structure, this indicates the cipher key and initialization vector will be either supplied in, or derived from, that context structure. In this case, the pointers to and the lengths of the cipher key and iv in the symOpData structure must be NULL and zero respectively.

Description: 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. It includes a structure to support a NIST 800-108 key derivation. This data structure is currently only used in chained usecases.

Additionally, if the cipher key is provided in the symOpData then the deriveCtxData fields must be set to NULL and zero.

See also:

CpaCySymPacketType

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

1.6.119 Typedef CpaCySymPacketType

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef enum *_CpaCySymPacketType* `CpaCySymPacketType`

Packet type for the cpaCySymPerformOp function

See also:

cpaCySymPerformOp()

Description: 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*.

1.6.120 Typedef CpaCySymSessionCtx

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef void *`CpaCySymSessionCtx`

Cryptographic component symmetric session context handle.

Description: 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.

1.6.121 Typedef CpaCySymSessionSetupData

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymSessionSetupData* `CpaCySymSessionSetupData`

Session Setup Data.

Description: 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.

1.6.122 Typedef CpaCySymSessionSetupData2

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymSessionSetupData2* `CpaCySymSessionSetupData2`

Session setup data with additional Crc Control Data

Description: This structure contains data relating to setting up a session and the crc control data used for data integrity computations. The client needs to complete the information in this structure in order to setup a session.

1.6.123 Typedef CpaCySymSessionUpdateData

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymSessionUpdateData* `CpaCySymSessionUpdateData`

Session Update Data.

Description: This structure contains data relating to resetting a session.

1.6.124 Typedef CpaCySymStats64

- Defined in file_api_lac_cpa_cy_sym.h

Typedef Documentation

typedef struct *_CpaCySymStats64* `CpaCySymStats64`

Cryptographic Component Statistics (64-bit version).

Description: 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.

1.6.125 Typedef CpaDcAutoSelectBest

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcAutoSelectBest* `CpaDcAutoSelectBest`

Supported modes for automatically selecting the best compression type.

When CPA_DC_ASB_ENABLED is used the output will be a format compliant block, whether the data is compressed or not.

Description: This enumeration lists the supported modes for automatically selecting the best encoding which would lead to the best compression results.

The following values are deprecated and should not be used. They will be removed in a future version of this file.

- CPA_DC_ASB_STATIC_DYNAMIC
- CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_STORED_HDRS
- CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_NO_HDRS

1.6.126 Typedef CpaDcCallbackFn

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef void (*`CpaDcCallbackFn`)(void *callbackTag, *CpaStatus* status)

Definition of callback function invoked for asynchronous cpaDc requests.

See also:

None

Description: This is the prototype for the cpaDc compression callback functions. The callback function is registered by the application using the *cpaDcInitSession()* 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

Note: None

Param callbackTag User-supplied value to help identify request.

Param status Status of the operation. Valid values are CPA_STATUS_SUCCESS, CPA_STATUS_FAIL and CPA_STATUS_UNSUPPORTED.

Retval None

Pre Component has been initialized.

Post None

1.6.127 Typedef CpaDcChainOpData

- Defined in file_api_dc_cpa_dc_chain.h

Typedef Documentation

typedef struct *_CpaDcChainOpData* `CpaDcChainOpData`

Compression chaining request input parameters.

Description: This structure contains the request information to use with compression chaining operations.

1.6.128 Typedef CpaDcChainOpData2

- Defined in file_api_dc_cpa_dc_chain.h

Typedef Documentation

typedef struct *_CpaDcChainOpData2* `CpaDcChainOpData2`

Chaining request op2 data for chained operations with optional verification step.

Description: This structure contains arguments for the cpaDcChainPerformOp2 API.

1.6.129 Typedef CpaDcChainOperations

- Defined in file_api_dc_cpa_dc_chain.h

Typedef Documentation

typedef enum *_CpaDcChainOperations* `CpaDcChainOperations`

Supported operations for compression chaining

Description: This enumeration lists the supported operations for compression chaining

1.6.130 Typedef CpaDcChainRqResults

- Defined in file_api_dc_cpa_dc_chain.h

Typedef Documentation

typedef struct *_CpaDcChainRqResults* `CpaDcChainRqResults`

Chaining request results data

Description: This structure contains the request results.

1.6.131 Typedef CpaDcChainRqVResults

- Defined in file_api_dc_cpa_dc_chain.h

Typedef Documentation

typedef struct *_CpaDcChainRqVResults* `CpaDcChainRqVResults`

Chaining request results data for chained operations with optional verification step.

Description: This structure contains the request results.

1.6.132 Typedef CpaDcChainSessionSetupData

- Defined in file_api_dc_cpa_dc_chain.h

Typedef Documentation

typedef struct *_CpaDcChainSessionSetupData* `CpaDcChainSessionSetupData`

Chaining Session Setup Data.

Description: This structure contains data relating to set up chaining sessions. The client needs to complete the information in this structure in order to setup chaining sessions.

1.6.133 Typedef CpaDcChainSessionSetupData2

- Defined in file_api_dc_cpa_dc_chain.h

Typedef Documentation

typedef struct *_CpaDcChainSessionSetupData2* `CpaDcChainSessionSetupData2`

Chaining session setup data with extended version of Dc and Cy Session setup data structure. Applications which wants to use the CpaDcSessionSetupData2 and CpaCySymSessionSetupData2 structures.

Description: This structure contains data relating to set up chaining sessions. The client needs to complete the information in this structure in order to setup chaining sessions.

1.6.134 Typedef CpaDcChainSessionType

- Defined in file_api_dc_cpa_dc_chain.h

Typedef Documentation

typedef enum *_CpaDcChainSessionType* `CpaDcChainSessionType`

Supported session types for data compression chaining.

Description: This enumeration lists the supported session types for data compression chaining.

1.6.135 Typedef CpaDcChainSubOpData2

- Defined in file_api_dc_cpa_dc_chain.h

Typedef Documentation

typedef struct *_CpaDcChainSubOpData2* `CpaDcChainSubOpData2`

Compression chaining request input parameters.

Description: This structure contains the request information to use with compression chaining sub-request operations.

1.6.136 Typedef CpaDcChecksum

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcChecksum* `CpaDcChecksum`

Supported checksum algorithms

Description: This enumeration lists the supported checksum algorithms Used to decide on file header and footer specifics.

1.6.137 Typedef CpaDcCompLvl

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcCompLvl* `CpaDcCompLvl`

Supported compression levels

Description: This enumerated lists the supported compressed levels. Lower values will result in less compressibility in less time.

1.6.138 Typedef CpaDcCompLZ4BlockMaxSize

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcCompLZ4BlockMaxSize* `CpaDcCompLZ4BlockMaxSize`

Maximum LZ4 output block size

Description: Maximum LZ4 output block size

1.6.139 Typedef CpaDcCompMinMatch

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcCompMinMatch* `CpaDcCompMinMatch`

Min match size in bytes

Description: This is the min match size that will be used for the search algorithm. It is only configurable for LZ4S.

1.6.140 Typedef CpaDcCompType

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcCompType* `CpaDcCompType`

Supported compression types

Description: This enumeration lists the supported data compression algorithms. In combination with CpaDcChecksum it is used to decide on the file header and footer format.

1.6.141 Typedef CpaDcCompWindowSize

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcCompWindowSize* `CpaDcCompWindowSize`

Support for defined algorithm window sizes

Description: This enumerated list defines the valid window sizes that can be used with the supported algorithms

1.6.142 Typedef CpaDcDir

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef *CpaDcSessionDir* `CpaDcDir`

1.6.143 Typedef CpaDcDpCallbackFn

- Defined in file_api_dc_cpa_dc_dp.h

Typedef Documentation

typedef void (*`CpaDcDpCallbackFn`)(*CpaDcDpOpData* *pOpData)

Definition of callback function for compression data plane API.

See also:

cpaDcDpRegCbFunc

Description: This is the callback function prototype. The callback function is registered by the application using the *cpaDcDpRegCbFunc* function call, and called back on completion of asynchronous requests made via calls to *cpaDcDpEnqueueOp* or cpaDcDpEnqueueOpBatch.

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

Note: None

Param pOpData [in] Pointer to the *CpaDcDpOpData* object which was supplied as part of the original request.

Return None

Pre Instance has been initialized. Callback has been registered with *cpaDcDpRegCbFunc*.

Post None

1.6.144 Typedef CpaDcDpOpData

- Defined in file_api_dc_cpa_dc_dp.h

Typedef Documentation

typedef struct *_CpaDcDpOpData* `CpaDcDpOpData`

Operation Data for compression data plane API.

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

Description: This structure contains data relating to a request to perform compression processing on one or more data buffers.

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

See also:

cpaDcDpEnqueueOp, cpaDcDpEnqueueOpBatch

1.6.145 Typedef CpaDcFlush

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcFlush* `CpaDcFlush`

Supported flush flags

Description: This enumerated list identifies the types of flush that can be specified for stateful and stateless cpaDcCompressData and cpaDcDecompressData functions.

1.6.146 Typedef CpaDcHuffType

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcHuffType* `CpaDcHuffType`

Supported Huffman Tree types

Selecting Full Dynamic Huffman trees generates compressed blocks with an RFC 1951 header specifying "compressed with dynamic Huffman codes". The headers are calculated on the data being compressed, requiring two passes.

Description: This enumeration lists support for Huffman Tree types. Selecting Static Huffman trees generates compressed blocks with an RFC 1951 header specifying "compressed with fixed Huffman trees".

Selecting Precompiled Huffman Trees generates blocks with RFC 1951 dynamic headers. The headers are pre-calculated and are specified by the file type.

1.6.147 Typedef CpaDcInstanceCapabilities

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaDcInstanceCapabilities* `CpaDcInstanceCapabilities`

Implementation Capabilities Structure

Description: This structure contains data relating to the capabilities of an implementation. The capabilities include supported compression algorithms, RFC 1951 options and whether the implementation supports both stateful and stateless compress and decompress sessions.

1.6.148 Typedef CpaDcInstanceNotificationCbFunc

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

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

Callback function for instance notification support.

See also:

cpaDcInstanceSetNotificationCb(),

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

Note: None

Param instanceHandle [in] Instance handle.

Param pCallbackTag [in] Opaque value provided by user while making individual function calls.

Param instanceEvent [in] The event that will trigger this function to get invoked.

Retval None

Pre Component has been initialized and the notification function has been set via the cpaDcInstanceSetNotificationCb function.

Post None

1.6.149 Typedef CpaDcIntegrityCrcSize

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcIntegrityCrcSize* `CpaDcIntegrityCrcSize`

Integrity CRC Size

Description: Enum of possible integrity CRC sizes.

1.6.150 Typedef CpaDcNsSetupData

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef *CpaDcSessionSetupData* `CpaDcNsSetupData`

1.6.151 Typedef CpaDcOpData

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaDcOpData* `CpaDcOpData`

(De)Compression request input parameters.

Description: This structure contains the request information for use with compression operations.

1.6.152 Typedef CpaDcOpData2

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaDcOpData2* `CpaDcOpData2`

(De)Compression request input parameters.

Description: This structure contains the request information for use with compression operations with additional options

1.6.153 Typedef CpaDcReqStatus

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcReqStatus* `CpaDcReqStatus`

Supported additional details from accelerator

Description: This enumeration lists the supported additional details from the accelerator. These may be useful in determining the best way to recover from a failure.

1.6.154 Typedef CpaDcRqResults

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaDcRqResults* `CpaDcRqResults`

Request results data

For stateful sessions the status, produced, consumed and endOfLastBlock results are per request values while the checksum value is cumulative across all requests on the session so far. In this case the checksum value is not guaranteed to be correct until the final compressed data has been processed.

Description: This structure contains the request results.

For stateless sessions, an initial checksum value is passed into the stateless operation. Once the stateless operation completes, the checksum value will contain checksum produced by the operation.

1.6.155 Typedef CpaDcSessionDir

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcSessionDir* `CpaDcSessionDir`

Supported session directions

Description: This enumerated list identifies the direction of a session. A session can be compress, decompress or both.

1.6.156 Typedef CpaDcSessionHandle

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef void *`CpaDcSessionHandle`

Compression API session handle type

Description: Handle used to uniquely identify a Compression API session handle. This handle is established upon registration with the API using *cpaDcInitSession()*.

1.6.157 Typedef CpaDcSessionSetupData

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaDcSessionSetupData* `CpaDcSessionSetupData`

Session Setup Data.

Description: 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.

1.6.158 Typedef CpaDcSessionSetupData2

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaDcSessionSetupData2* `CpaDcSessionSetupData2`

Session Setup Data with additional Crc Control Data

Description: This structure contains data relating to setting up a session and the crc control data used for data integrity computations. The client needs to complete the information in this structure in order to setup a session.

1.6.159 Typedef CpaDcSessionState

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcSessionState* `CpaDcSessionState`

Supported session state settings

Stateful sessions are limited to have only one in-flight message per session. This means a compress or decompress request must be complete before a new request can be started. This applies equally to sessions that are uni-directional in nature and sessions that are combined compress and decompress. Completion occurs when the synchronous function returns, or when the asynchronous callback function has completed.

Description: This enumerated list identifies the stateful setting of a session. A session can be either stateful or stateless.

1.6.160 Typedef CpaDcSessionUpdateData

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaDcSessionUpdateData* `CpaDcSessionUpdateData`

Session Update Data.

Description: This structure contains data relating to updating up a session. The client needs to complete the information in this structure in order to update a session.

1.6.161 Typedef CpaDcSkipData

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaDcSkipData* `CpaDcSkipData`

Skip Region Data.

Description: This structure contains data relating to configuring skip region behaviour. A skip region is a region of an input buffer that should be omitted from processing or a region that should be inserted into the output buffer.

1.6.162 Typedef CpaDcSkipMode

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef enum *_CpaDcSkipMode* `CpaDcSkipMode`

Supported modes for skipping regions of input or output buffers.

Description: This enumeration lists the supported modes for skipping regions of input or output buffers.

1.6.163 Typedef CpaDcState

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef *CpaDcSessionState* `CpaDcState`

1.6.164 Typedef CpaDcStats

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaDcStats* `CpaDcStats`

Compression Statistics Data.

Description: This structure contains data elements corresponding to statistics. Statistics are collected on a per instance basis and include: jobs submitted and completed for both compression and decompression.

1.6.165 Typedef CpaFlatBuffer

- Defined in file_api_cpa.h

Typedef Documentation

typedef struct *_CpaFlatBuffer* `CpaFlatBuffer`

Flat buffer structure containing a pointer and length member.

Description: 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*.

1.6.166 Typedef CpaInstanceAllocPolicy

- Defined in file_api_cpa.h

Typedef Documentation

typedef enum *_CpaInstanceAllocPolicy* `CpaInstanceAllocPolicy`

Instance Allocation Policies

Description: Enumeration of the possible instance allocation policies that may be used for allocating instances using the *cpaAllocInstance()* API.

1.6.167 Typedef CpaInstanceEvent

- Defined in file_api_cpa.h

Typedef Documentation

typedef enum *_CpaInstanceEvent* `CpaInstanceEvent`

Instance Events

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

1.6.168 Typedef CpaInstanceHandle

- Defined in file_api_cpa.h

Typedef Documentation

typedef void *`CpaInstanceHandle`

Instance handle type.

Description: Handle used to uniquely identify an instance.

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

1.6.169 Typedef CpaInstanceInfo2

- Defined in file_api_cpa.h

Typedef Documentation

typedef struct *_CpaInstanceInfo2* `CpaInstanceInfo2`

Instance Info Structure, version 2

Description: Structure that contains the information to describe the instance.

1.6.170 Typedef CpaIntegrityCrc

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaIntegrityCrc* `CpaIntegrityCrc`

Integrity CRC calculation details

Description: This structure contains information about resulting integrity CRC calculations performed for a single request.

1.6.171 Typedef CpaIntegrityCrc64b

- Defined in file_api_dc_cpa_dc.h

Typedef Documentation

typedef struct *_CpaIntegrityCrc64b* `CpaIntegrityCrc64b`

Integrity CRC64 calculation details

Description: This structure contains information about resulting integrity CRC64 calculations performed for a single request.

1.6.172 Typedef CpaOperationalState

- Defined in file_api_cpa.h

Typedef Documentation

typedef enum *_CpaOperationalState* `CpaOperationalState`

Instance operational state

Description: Enumeration of the different operational states that are possible.

1.6.173 Typedef CpaPhysBufferList

- Defined in file_api_cpa.h

Typedef Documentation

typedef struct *_CpaPhysBufferList* `CpaPhysBufferList`

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

Description: 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.

1.6.174 Typedef CpaPhysFlatBuffer

- Defined in file_api_cpa.h

Typedef Documentation

typedef struct *_CpaPhysFlatBuffer* `CpaPhysFlatBuffer`

Flat buffer structure with physical address.

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

1.6.175 Typedef CpaPhysicalAddr

- Defined in file_api_cpa.h

Typedef Documentation

typedef *Cpa64U* `CpaPhysicalAddr`

Physical memory address.

Description: Type for physical memory addresses.

1.6.176 Typedef CpaPhysicalInstanceId

- Defined in file_api_cpa.h

Typedef Documentation

typedef struct *_CpaPhysicalInstanceId* `CpaPhysicalInstanceId`

Physical Instance ID

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.

Description: Identifies the physical instance of an accelerator execution engine.

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.

1.6.177 Typedef CpaStatus

- Defined in file_api_cpa.h

Typedef Documentation

typedef *Cpa32S* `CpaStatus`

API status value type definition

Description: 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.

1.6.178 Typedef CpaVirtualToPhysical

- Defined in file_api_cpa.h

Typedef Documentation

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

Virtual to physical address conversion routine.

See also:

None

Description: 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

Param pVirtualAddr [in] Virtual address to be converted.

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

Post None

2 Revision History

2.1 Cryptographic API Reference Revision History

Date	Revision	Description
Apr 2022	009	Changed version of the crypto API to v3.0. Added RSA8K support.
May 2021	008	Changed version of the crypto API to v2.5. Added support for SM2.
Nov 2020	007	Changed version of the crypto API to v2.4. Added support for ChaCha20-Poly1305. Added support for SM4 in ECB, CBC and CTR modes. Added support for SM3. Added support for SHA3-224, SHA3-384 and SHA3-512.
Mar 2020	006	Added HKDF API Added 22519 and 448 curve support to cpa_cy_ec.h
Apr 2018	005	Added session update API
Jul 2016	004	Added Intel Key Protection Technology (KPT) API
Oct 2015	003	Changed version of the crypto API for v2.0. Added ZUC-EEA3 and ZUC-EIA3 support Added SHA3-256 support
Sep 2015	002	Incrementing crypto API versio to v1.9. Adding CPA_STATUS_UNSUPPORTED as a return status.
Jun 2014	001	First "public" version of the document. Based on "Intel Confidential" document number 410923-1.8.

2.2 Compression API Reference Revision History

Date	Revision	Description
Apr 2022	014	Compression API v3.1. Removing deprecated types, CpaDcFileType. Added support for LZ4 and LZ4s. Added support for sessionless (Ns) functions.
Jul 2021	013	Compression API v2.7. Added CPA_DC_ASB_ENABLED. Added a flag to notify of an uncompressed block in CpaDcRqResults.
Nov 2020	012	Compression API v2.6. Added support for integrity CRCs.
Sep 2020	011	Compression API v2.5. Added cpaDcDeflateCompressBound()
Jun 2020	010	Compression API v2.4. Added cpaDcUpdateSession()
Jan 2019	009	Compression API v2.3. Added chaining support for compression & hash operations.

continues on next page

Table 2 – continued from previous page

Date	Revision	Description
Apr 2018	008	Compression API v2.2. Added support for compressAndVerifyAndRecover
Feb 2018	007	Adding support for Compress and Verify strict mode
Jun 2016	006	Compression API v2.1. Added support for Compress and Verify Added support for Batch and Pack Compression
Oct 2015	005	Compression API v2.0. Added new error codes to CpaDcReqStatus in cpa_dc.h
Sep 2015	004	Compression API v1.6. Added CPA_STATUS_UNSUPPORTED as a return status Deprecating use of fileType and deflateWindowSize fields from CpaDcSessionSetupData in cpa_dc.h Clarifying documentation for srcBufferLen, bufferLenToCompress, destBufferLen and bufferLenForData in CpaDcDpOpData in cpa_dc_dp.h
Aug 2015	003	Compression API v1.5. Adding reportParityError to the DC instance capabilities.
Oct 2014	002	Adding cpaDcResetSession API.
Jun 2014	001	First "public" version of the document. Based on "Intel Confidential" document number 410926-1.3.

Index