



# **Intel® EP80579 Software for Security Applications on Intel® QuickAssist Technology Cryptographic API Reference**

*Automatically generated from sources, May 19, 2009.*

Reference Number: 320184, Revision -003

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

This document contains information on products in the design phase of development.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See [http://www.intel.com/products/processor\\_number](http://www.intel.com/products/processor_number) for details.

Code Names are only for use by Intel to identify products, platforms, programs, services, etc. ("products") in development by Intel that have not been made commercially available to the public, i.e., announced, launched or shipped. They are never to be used as "commercial" names for products. Also, they are not intended to function as trademarks.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © Intel Corporation 2009. All Rights Reserved.

## Revision History

Date	Revision	Description
May 2009	-003	All cpaCy*QueryStats functions now document the return value of CPA_STATUS_RESOURCE Documents version 1.1.1 of the API.
November 2008	-002	FreeBSD support added. There are no API changes. Documents version 1.1.1 of the API.
July 2008	-001	First released version of this document. Documents version 1.1 of the API.

# Table of Contents

<b>1 CPA API.....</b>	<b>1</b>
1.1 Detailed Description.....	1
1.2 Modules.....	1
<b>2 Base Data Types [CPA API].....</b>	<b>2</b>
2.1 Detailed Description.....	2
2.2 Data Structures.....	2
2.3 Defines.....	2
2.4 Typedefs.....	3
2.5 Enumerations.....	3
2.6 Data Structure Documentation.....	3
2.6.1 _CpaFlatBuffer Struct Reference.....	3
2.6.2 _CpaBufferList Struct Reference.....	4
2.6.3 _CpaInstanceInfo Struct Reference.....	5
2.7 Define Documentation.....	6
2.8 Typedef Documentation.....	7
2.9 Enumeration Type Documentation.....	8
<b>3 CPA Type Definition [CPA API].....</b>	<b>10</b>
3.1 Detailed Description.....	10
3.2 Defines.....	10
3.3 Typedefs.....	10
3.4 Enumerations.....	10
3.5 Define Documentation.....	10
3.6 Typedef Documentation.....	11
3.7 Enumeration Type Documentation.....	11
<b>4 Cryptographic API. [CPA API].....</b>	<b>12</b>
4.1 Detailed Description.....	12
4.2 Modules.....	12
<b>5 Cryptographic Common API. [Cryptographic API.].....</b>	<b>14</b>
5.1 Detailed Description.....	14
5.2 Typedefs.....	14
5.3 Enumerations.....	14
5.4 Functions.....	14
5.5 Typedef Documentation.....	15
5.6 Enumeration Type Documentation.....	17
5.7 Function Documentation.....	17
<b>6 Public Key Encryption Diffie-Hellman API. [Cryptographic API.].....</b>	<b>24</b>
6.1 Detailed Description.....	24
6.2 Data Structures.....	24
6.3 Typedefs.....	24
6.4 Functions.....	24
6.5 Data Structure Documentation.....	25
6.5.1 _CpaCyDhPhase1KeyGenOpData Struct Reference.....	25
6.5.2 _CpaCyDhPhase2SecretKeyGenOpData Struct Reference.....	26
6.5.3 _CpaCyDhStats Struct Reference.....	27
6.6 Typedef Documentation.....	28
6.7 Function Documentation.....	29
<b>7 Public Key Encryption DSA API. [Cryptographic API.].....</b>	<b>34</b>
7.1 Detailed Description.....	34
7.2 Data Structures.....	34

# Table of Contents

<b>7 Public Key Encryption DSA API. [Cryptographic API.]</b>	
7.3 Typedefs.....	34
7.4 Functions.....	35
7.5 Data Structure Documentation.....	35
7.5.1 _CpaCyDsaPParamGenOpData Struct Reference.....	36
7.5.2 _CpaCyDsaGParamGenOpData Struct Reference.....	37
7.5.3 _CpaCyDsaYParamGenOpData Struct Reference.....	38
7.5.4 _CpaCyDsaRSignOpData Struct Reference.....	39
7.5.5 _CpaCyDsaSSignOpData Struct Reference.....	41
7.5.6 _CpaCyDsaRSSignOpData Struct Reference.....	42
7.5.7 _CpaCyDsaVerifyOpData Struct Reference.....	44
7.5.8 _CpaCyDsaStats Struct Reference.....	45
7.6 Typedef Documentation.....	49
7.7 Function Documentation.....	54
<b>8 Crypto Instance Maintenance API. [Cryptographic API.]</b>	<b>65</b>
8.1 Detailed Description.....	65
8.2 Functions.....	65
8.3 Function Documentation.....	65
<b>9 Key and Mask Generation API. [Cryptographic API.]</b>	<b>68</b>
9.1 Detailed Description.....	68
9.2 Data Structures.....	68
9.3 Defines.....	68
9.4 Typedefs.....	68
9.5 Enumerations.....	68
9.6 Functions.....	69
9.7 Data Structure Documentation.....	69
9.7.1 _CpaCyKeyGenSslOpData Struct Reference.....	69
9.7.2 _CpaCyKeyGenTlsOpData Struct Reference.....	71
9.7.3 _CpaCyKeyGenMgfOpData Struct Reference.....	73
9.7.4 _CpaCyKeyGenStats Struct Reference.....	74
9.8 Define Documentation.....	75
9.9 Typedef Documentation.....	76
9.10 Enumeration Type Documentation.....	77
9.11 Function Documentation.....	77
<b>10 Crypto API Large Number. [Cryptographic API.]</b>	<b>83</b>
10.1 Detailed Description.....	83
10.2 Data Structures.....	83
10.3 Typedefs.....	83
10.4 Functions.....	83
10.5 Data Structure Documentation.....	83
10.5.1 _CpaCyLnModExpOpData Struct Reference.....	84
10.5.2 _CpaCyLnModInvOpData Struct Reference.....	85
10.5.3 _CpaCyLnStats Struct Reference.....	86
10.6 Typedef Documentation.....	87
10.7 Function Documentation.....	88
<b>11 Prime Number Test API. [Cryptographic API.]</b>	<b>92</b>
11.1 Detailed Description.....	92
11.2 Data Structures.....	92
11.3 Typedefs.....	92
11.4 Functions.....	92
11.5 Data Structure Documentation.....	92

# Table of Contents

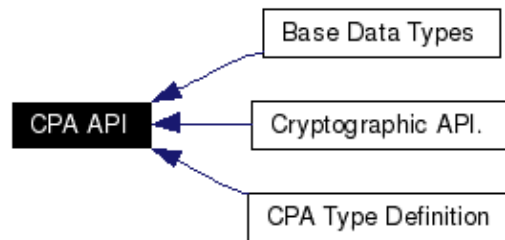
<b>11 Prime Number Test API. [Cryptographic API.]</b>	
11.5.1 _CpaCyPrimeTestOpData Struct Reference.....	92
11.5.2 _CpaCyPrimeStats Struct Reference.....	94
11.6 Typedef Documentation.....	95
11.7 Function Documentation.....	96
<b>12 Random Bit/Number Generation API. [Cryptographic API.]</b>	<b>99</b>
12.1 Detailed Description.....	99
12.2 Data Structures.....	99
12.3 Defines.....	99
12.4 Typedefs.....	99
12.5 Functions.....	99
12.6 Data Structure Documentation.....	100
12.6.1 _CpaCyRandStats Struct Reference.....	100
12.6.2 _CpaCyRandGenOpData Struct Reference.....	101
12.6.3 _CpaCyRandSeedOpData Struct Reference.....	102
12.7 Define Documentation.....	103
12.8 Typedef Documentation.....	103
12.9 Function Documentation.....	104
<b>13 Public Key Encryption RSA API. [Cryptographic API.]</b>	<b>108</b>
13.1 Detailed Description.....	108
13.2 Data Structures.....	108
13.3 Typedefs.....	108
13.4 Enumerations.....	109
13.5 Functions.....	109
13.6 Data Structure Documentation.....	109
13.6.1 _CpaCyRsaPublicKey Struct Reference.....	109
13.6.2 _CpaCyRsaPrivateKeyRep1 Struct Reference.....	111
13.6.3 _CpaCyRsaPrivateKeyRep2 Struct Reference.....	112
13.6.4 _CpaCyRsaPrivateKey Struct Reference.....	113
13.6.5 _CpaCyRsaKeyGenOpData Struct Reference.....	115
13.6.6 _CpaCyRsaEncryptOpData Struct Reference.....	116
13.6.7 _CpaCyRsaDecryptOpData Struct Reference.....	118
13.6.8 _CpaCyRsaStats Struct Reference.....	120
13.7 Typedef Documentation.....	122
13.8 Enumeration Type Documentation.....	124
13.9 Function Documentation.....	125
<b>14 Symmetric Cipher and Hash Crypto API [Cryptographic API.]</b>	<b>131</b>
14.1 Detailed Description.....	131
14.2 Data Structures.....	131
14.3 Typedefs.....	131
14.4 Enumerations.....	132
14.5 Functions.....	133
14.6 Data Structure Documentation.....	133
14.6.1 _CpaCySymCipherSetupData Struct Reference.....	133
14.6.2 _CpaCySymHashNestedModeSetupData Struct Reference.....	134
14.6.3 _CpaCySymHashAuthModeSetupData Struct Reference.....	135
14.6.4 _CpaCySymHashSetupData Struct Reference.....	136
14.6.5 _CpaCySymSessionSetupData Struct Reference.....	137
14.6.6 _CpaCySymOpData Struct Reference.....	139
14.6.7 _CpaCySymStats Struct Reference.....	141
14.7 Typedef Documentation.....	143
14.8 Enumeration Type Documentation.....	146

# Table of Contents

<b>14 Symmetric Cipher and Hash Crypto API [Cryptographic API.]</b>	
14.9 Function Documentation.....	149

# 1 CPA API

Collaboration diagram for CPA API:



## 1.1 Detailed Description

This is the top level API definition.

It contains structures, data types and definitions that are common across the interface.

## 1.2 Modules

- **Base Data Types**  
The base data types for the Intel CPA API.
- **CPA Type Definition**  
This is the CPA Type Definitions.
- **Cryptographic API.**  
These functions specify the Cryptographic API.



# 2 Base Data Types

## [CPA API]

Collaboration diagram for Base Data Types:



## 2.1 Detailed Description

The base data types for the Intel CPA API.

## 2.2 Data Structures

- struct **\_CpaFlatBuffer**  
Flat buffer structure containing a pointer and length member.
- struct **\_CpaBufferList**  
Scatter/Gather buffer list containing an array of Simple buffers.
- struct **\_CpaInstanceInfo**  
Instance Info Structure.

## 2.3 Defines

- #define **CPA\_INSTANCE\_HANDLE\_SINGLE**  
Default instantiation handle value where there is only a single instance.
- #define **CPA\_STATUS\_SUCCESS**  
Success status value.
- #define **CPA\_STATUS\_FAIL**  
Fail status value.
- #define **CPA\_STATUS\_RETRY**  
Retry status value.
- #define **CPA\_STATUS\_RESOURCE**  
The resource that has been requested is unavailable.
- #define **CPA\_STATUS\_INVALID\_PARAM**  
Invalid parameter has been passed in.
- #define **CPA\_STATUS\_FATAL**  
A serious error has occurred.
- #define **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).
- #define **CPA\_STATUS\_STR\_SUCCESS**  
Status string for **CPA\_STATUS\_SUCCESS**.
- #define **CPA\_STATUS\_STR\_FAIL**  
Status string for **CPA\_STATUS\_FAIL**.
- #define **CPA\_STATUS\_STR\_RETRY**  
Status string for **CPA\_STATUS\_RETRY**.
- #define **CPA\_STATUS\_STR\_RESOURCE**  
Status string for **CPA\_STATUS\_RESOURCE**.
- #define **CPA\_STATUS\_STR\_INVALID\_PARAM**  
Status string for **CPA\_STATUS\_INVALID\_PARAM**.
- #define **CPA\_STATUS\_STR\_FATAL**

## 2.3 Defines

- Status string for **CPA\_STATUS\_FATAL**.
- **#define CPA\_INSTANCE\_MAX\_NAME\_SIZE\_IN\_BYTES**  
Instance Info Max string lengths Maximum instance info name string length in bytes.
- **#define CPA\_INSTANCE\_MAX\_VERSION\_SIZE\_IN\_BYTES**  
Maximum instance info version string length in bytes.

## 2.4 Typedefs

- **typedef void \* CpaInstanceHandle**  
Instance handle type.
- **typedef \_CpaFlatBuffer CpaFlatBuffer**  
Flat buffer structure containing a pointer and length member.
- **typedef \_CpaBufferList CpaBufferList**  
Scatter/Gather buffer list containing an array of Simple buffers.
- **typedef Cpa32S CpaStatus**  
API status value type definition.
- **typedef enum \_CpaInstanceType CpaInstanceType**  
Instance Types.
- **typedef enum \_CpaInstanceState CpaInstanceState**  
Instance State.
- **typedef \_CpaInstanceInfo CpaInstanceInfo**  
Instance Info Structure.
- **typedef enum \_CpaInstanceEvent CpaInstanceEvent**  
Instance Events.

## 2.5 Enumerations

- **enum \_CpaInstanceType {**  
    **CPA\_INSTANCE\_TYPE\_CRYPTO,**  
    **CPA\_INSTANCE\_TYPE\_DATA\_COMPRESSION,**  
    **CPA\_INSTANCE\_TYPE\_RAID,**  
    **CPA\_INSTANCE\_TYPE\_XML,**  
    **CPA\_INSTANCE\_TYPE\_REGEX**  
    **}**  
    Instance Types.
- **enum \_CpaInstanceState {**  
    **CPA\_INSTANCE\_STATE\_INITIALISED,**  
    **CPA\_INSTANCE\_STATE\_SHUTDOWN**  
    **}**  
    Instance State.
- **enum \_CpaInstanceEvent {**  
    **CPA\_INSTANCE\_EVENT\_CREATION,**  
    **CPA\_INSTANCE\_EVENT\_DELETION**  
    **}**  
    Instance Events.

---

## 2.6 Data Structure Documentation

### 2.6.1 \_CpaFlatBuffer Struct Reference

## 2.6.1 \_CpaFlatBuffer Struct Reference

### 2.6.1.1 Detailed Description

Flat buffer structure containing a pointer and length member.

A flat buffer structure. The data pointer, pData, is a virtual address however the actual data pointed to is required to be in contiguous physical memory. It is expected that this buffer handle will be used when simple, unchained buffers are needed.

### 2.6.1.2 Data Fields

- **Cpa32U dataLenInBytes**  
Data length specified in bytes.
- **Cpa8U \* pData**  
The data pointer is a virtual address, however the actual data pointed to is required to be in contiguous physical memory.

### 2.6.1.3 Field Documentation

#### Cpa32U \_CpaFlatBuffer::dataLenInBytes

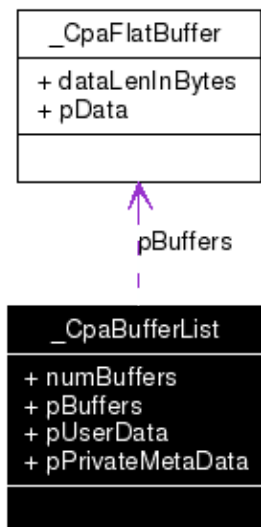
Data length specified in bytes.

#### Cpa8U\* \_CpaFlatBuffer::pData

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

## 2.6.2 \_CpaBufferList Struct Reference

Collaboration diagram for \_CpaBufferList:



### 2.6.2.1 Detailed Description

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

A Scatter/Gather buffer list structure. It is expected that this buffer structure will be used where more than one flat buffer can be provided on an particular API.

## 2.6.2 \_CpaBufferList Struct Reference

**IMPORTANT** - The memory for the pPrivateMetaData member must be allocated by the client as contiguous memory. When allocating memory for pPrivateMetaData a call to 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.

### 2.6.2.2 Data Fields

- **Cpa32U numBuffers**  
Number of pointers.
- **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 Meta representation of this buffer List - the memory for this buffer needs to be allocated by the client as contiguous data.

### 2.6.2.3 Field Documentation

#### **Cpa32U \_CpaBufferList::numBuffers**

Number of pointers.

#### **CpaFlatBuffer\* \_CpaBufferList::pBuffers**

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

#### **void\* \_CpaBufferList::pUserData**

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

#### **void\* \_CpaBufferList::pPrivateMetaData**

Private Meta 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 cpaCyBufferListGetMetaSize. If cpaCyBufferListGetMetaSize returns a size of zero no memory needs to be allocated, and this parameter can be NULL.

---

## 2.6.3 \_CpaInstanceInfo Struct Reference

### 2.6.3.1 Detailed Description

Instance Info Structure.

Structure that contains the information to describe the instance.

### 2.6.3.2 Data Fields

- **CpaInstanceType type**  
Type definition for this instance.
- **CpaInstanceState state**  
Operational state of the instance.
- **Cpa8U name [CPA\_INSTANCE\_MAX\_NAME\_SIZE\_IN\_BYTES]**

### 2.6.3 \_CpaInstanceInfo Struct Reference

Simple text string identifier for the instance.

- **Cpa8U version** [CPA\_INSTANCE\_MAX\_VERSION\_SIZE\_IN\_BYTES]  
Version string.

#### 2.6.3.3 Field Documentation

##### **CpaInstanceType \_CpaInstanceInfo::type**

Type definition for this instance.

##### **CpaInstanceState \_CpaInstanceInfo::state**

Operational state of the instance.

##### **Cpa8U \_CpaInstanceInfo::name**[CPA\_INSTANCE\_MAX\_NAME\_SIZE\_IN\_BYTES]

Simple text string identifier for the instance.

##### **Cpa8U \_CpaInstanceInfo::version**[CPA\_INSTANCE\_MAX\_VERSION\_SIZE\_IN\_BYTES]

Version string.

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

---

## 2.7 Define Documentation

##### **#define CPA\_INSTANCE\_HANDLE\_SINGLE**

Default instantiation handle value where there is only a single instance.

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

##### **#define CPA\_STATUS\_SUCCESS**

Success status value.

##### **#define CPA\_STATUS\_FAIL**

Fail status value.

##### **#define CPA\_STATUS\_RETRY**

Retry status value.

##### **#define CPA\_STATUS\_RESOURCE**

The resource that has been requested is unavailable.

Refer to relevant sections of the API for specifics on what the suggested course of action is.

##### **#define CPA\_STATUS\_INVALID\_PARAM**

Invalid parameter has been passed in.

##### **#define CPA\_STATUS\_FATAL**

A serious error has occurred.

Recommended course of action is to shutdown and restart the component.

## 2.7 Define Documentation

```
#define 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.

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**.

```
#define CPA_STATUS_STR_SUCCESS
```

Status string for **CPA\_STATUS\_SUCCESS**.

```
#define CPA_STATUS_STR_FAIL
```

Status string for **CPA\_STATUS\_FAIL**.

```
#define CPA_STATUS_STR_RETRY
```

Status string for **CPA\_STATUS\_RETRY**.

```
#define CPA_STATUS_STR_RESOURCE
```

Status string for **CPA\_STATUS\_RESOURCE**.

```
#define CPA_STATUS_STR_INVALID_PARAM
```

Status string for **CPA\_STATUS\_INVALID\_PARAM**.

```
#define CPA_STATUS_STR_FATAL
```

Status string for **CPA\_STATUS\_FATAL**.

```
#define CPA_INSTANCE_MAX_NAME_SIZE_IN_BYTES
```

Instance Info Max string lengths Maximum instance info name string length in bytes.

Definitions of the instance info max string lengths.

```
#define CPA_INSTANCE_MAX_VERSION_SIZE_IN_BYTES
```

Maximum instance info version string length in bytes.

---

## 2.8 Typedef Documentation

```
typedef void* CpaInstanceHandle
```

Instance handle type.

Handle used to uniquely identify an instance.

**Note:**

Where only a single instantiation exists this field must be set to **CPA\_INSTANCE\_HANDLE\_DEFAULT**.

```
typedef struct _CpaFlatBuffer CpaFlatBuffer
```

Flat buffer structure containing a pointer and length member.

A flat buffer structure. The data pointer, pData, is a virtual address however the actual data pointed to is required to be in contiguous physical memory. It is expected that this buffer handle will be used when simple, unchained buffers are needed.

## 2.8 Typedef Documentation

typedef struct **\_CpaBufferList CpaBufferList**

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

A Scatter/Gather buffer list structure. It is expected that this buffer structure will be used where more than one flat buffer can be provided on an particular API.

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

typedef **Cpa32S CpaStatus**

API status value type definition.

This type definition is used for the return values used in all the API functions. Common values are defined, for example see **CPA\_STATUS\_SUCCESS**, **CPA\_STATUS\_FAIL**, etc.

typedef enum **\_CpaInstanceType CpaInstanceType**

Instance Types.

Enumeration of the different instance types.

typedef enum **\_CpaInstanceState CpaInstanceState**

Instance State.

Enumeration of the different instance states that are possible.

typedef struct **\_CpaInstanceInfo CpaInstanceInfo**

Instance Info Structure.

Structure that contains the information to describe the instance.

typedef enum **\_CpaInstanceEvent CpaInstanceEvent**

Instance Events.

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

---

## 2.9 Enumeration Type Documentation

enum **\_CpaInstanceType**

Instance Types.

Enumeration of the different instance types.

### Enumerator:

<i>CPA_INSTANCE_TYPE_CRYPTO</i>	Cryptographic instance type.
<i>CPA_INSTANCE_TYPE_DATA_COMPRESSION</i>	Data compression instance type.
<i>CPA_INSTANCE_TYPE_RAID</i>	RAID instance type.
<i>CPA_INSTANCE_TYPE_XML</i>	XML instance type.

## 2.9 Enumeration Type Documentation

*CPA\_INSTANCE\_TYPE\_REGEX*

Regular Expression  
instance type.

### enum **\_CpaInstanceState**

Instance State.

Enumeration of the different instance states that are possible.

#### **Enumerator:**

*CPA\_INSTANCE\_STATE\_INITIALISED* Instance is in the initialized state and ready for use.

*CPA\_INSTANCE\_STATE\_SHUTDOWN* Instance is in the shutdown state and not available for use.

### enum **\_CpaInstanceEvent**

Instance Events.

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

#### **Enumerator:**

*CPA\_INSTANCE\_EVENT\_CREATION* Event type that triggers the registered instance notification callback function when an instance is created.

*CPA\_INSTANCE\_EVENT\_DELETION* Event type that triggers the registered instance notification callback function when an instance is deleted.



# 3 CPA Type Definition

## [CPA API]

Collaboration diagram for CPA Type Definition:



### 3.1 Detailed Description

This is the CPA Type Definitions.

### 3.2 Defines

- **#define NULL**  
NULL definition.
- **#define TRUE**  
True value definition.
- **#define FALSE**  
False value definition.

### 3.3 Typedefs

- **typedef enum \_CpaBoolean CpaBoolean**  
Boolean type.

### 3.4 Enumerations

- **enum \_CpaBoolean {**  
    **CPA\_FALSE,**  
    **CPA\_TRUE**  
    **}**  
Boolean type.

### 3.5 Define Documentation

```
#define NULL  
NULL definition.
```

```
#define TRUE  
True value definition.
```

```
#define FALSE  
False value definition.
```

## 3.6 Typedef Documentation

```
typedef enum _CpaBoolean CpaBoolean  
    Boolean type.
```

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

---

## 3.7 Enumeration Type Documentation

```
enum _CpaBoolean  
    Boolean type.
```

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

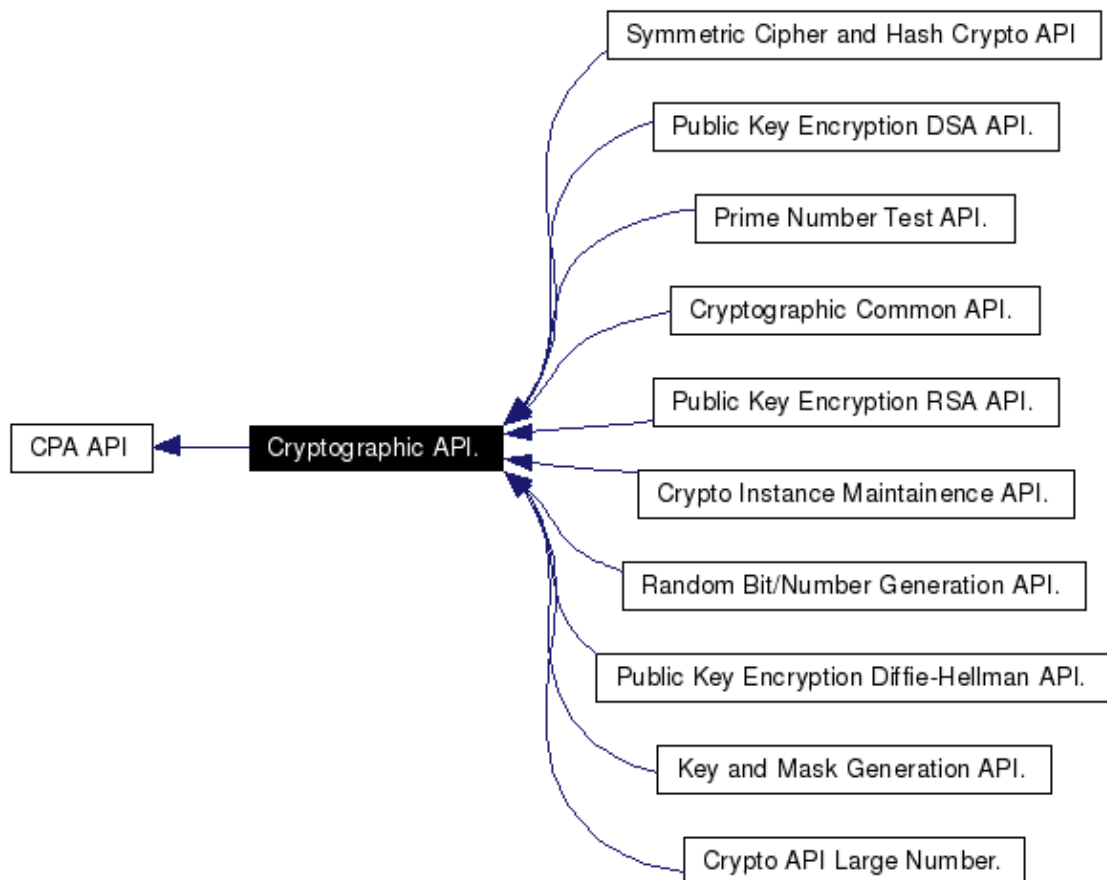
**Enumerator:**

*CPA\_FALSE* False value.  
*CPA\_TRUE* True value.

## 4 Cryptographic API.

### [CPA API]

Collaboration diagram for Cryptographic API.:



### 4.1 Detailed Description

These functions specify the Cryptographic API.

### 4.2 Modules

- **Cryptographic Common API.**

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

- **Public Key Encryption Diffie-Hellman API.**

These functions specify the API for Public Key Encryption (cryptography) operations for use with Diffie-Hellman algorithm.

- **Public Key Encryption DSA API.**

These functions specify the API for Public Key Encryption (cryptography) Digital Signature Algorithm (DSA) operations. The FIPS PUB 186-2 with Change Notice 1 specification is supported.

- **Crypto Instance Maintenance API.**

## 4.2 Modules

These functions specify the Instance Maintenance API for available Crypto Instances.

- **Key and Mask Generation API.**

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

- **Crypto API Large Number.**

These functions specify the Cryptographic API for Large Number Operations.

- **Prime Number Test API.**

These functions specify the API for the prime number test operations.

- **Random Bit/Number Generation API.**

These functions specify the API for the Cryptographic Random Bit and Random number generation.

- **Public Key Encryption RSA API.**

These functions specify the API for Public Key Encryption (cryptography) RSA operations.

- **Symmetric Cipher and Hash Crypto API**

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

# 5 Cryptographic Common API.

## [Cryptographic API.]

Collaboration diagram for Cryptographic Common API.:



## 5.1 Detailed Description

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

## 5.2 Typedefs

- typedef void(\* **CpaCyGenericCbFunc** )(void \*pCallbackTag, **CpaStatus** status, void \*pOpData)  
Definition of the crypto generic callback function.
- typedef void(\* **CpaCyGenFlatBufCbFunc** )(void \*pCallbackTag, **CpaStatus** status, void \*pOpdata, **CpaFlatBuffer** \*pOut)  
Definition of generic callback function with an additional output CpaFlatBuffer parameter.
- typedef void(\* **CpaCyInstanceNotificationCbFunc** )(void \*pCallbackTag, const **CpaInstanceEvent** instanceEvent)  
Callback function for instance notification support.

## 5.3 Enumerations

- enum **CpaCyPriority** {  
    **CPA\_CY\_PRIORITY\_NORMAL**,  
    **CPA\_CY\_PRIORITY\_HIGH**  
}
- Request priority.

## 5.4 Functions

- **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.
- **CpaStatus cpaCyGetStatusText** (const **CpaInstanceHandle** instanceHandle, **CpaStatus** errStatus, **Cpa8S** \*pStatusText)  
Function to return a string indicating the specific error that occurred for a particular instance.
- **CpaStatus cpaCyGetNumInstances** (**Cpa16U** \*pNumInstances)  
Get the number of instances that are supported by the API implementation.
- **CpaStatus cpaCyGetInstances** (**Cpa16U** numInstances, **CpaInstanceHandle** \*cyInstances)  
Get the handles to the instances that are supported by the API implementation.
- **CpaStatus cpaCyInstanceGetInfo** (const **CpaInstanceHandle** instanceHandle, **CpaInstanceInfo** \*pInstanceInfo)  
Function to get information on a particular instance.
- **CpaStatus cpaCyInstanceSetNotificationCb** (const **CpaCyInstanceNotificationCbFunc** pInstanceNotificationCb, void \*pCallbackTag)

## 5.5 Typedef Documentation

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

Definition of the crypto generic callback function.

This data structure specifies the prototype for a generic callback function

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

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

**Return values:**

None

**Precondition:**

Component has been initialized.

**Postcondition:**

None

**Note:**

None

**See also:**

**cpaCyKeyGenSsl()**

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

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

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

**Context:**

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

## 5.5 Typedef Documentation

**Assumptions:**

None

**Side-Effects:**

None

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

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

**Return values:**

*None*

**Precondition:**

Component has been initialized.

**Postcondition:**

None

**Note:**

None

**See also:**

None

```
typedef void(* CpaCylInstanceNotificationCbFunc)(void *pCallbackTag, const CpaInstanceEvent instanceEvent)
```

Callback function for instance notification support.

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

## 5.6 Enumeration Type Documentation

### Thread-safe:

Yes

### Parameters:

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

### Return values:

None

### Precondition:

Component has been initialized and the notification function has been set via the `cpaInstanceSetNotificationCb` function.

### Postcondition:

None

### Note:

None

### See also:

`cpaInstanceSetNotificationCb()`,

---

## 5.6 Enumeration Type Documentation

### enum **CpaCyPriority**

Request priority.

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

### Enumerator:

- CPA\_CY\_PRIORITY\_NORMAL* Normal priority.
- CPA\_CY\_PRIORITY\_HIGH* High priority.

---

## 5.7 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`.

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.



## 5.7 Function Documentation

### Assumptions:

None

### Side-Effects:

None

### Blocking:

No

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

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

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.

### Precondition:

None.

### Postcondition:

None

### Note:

None

### See also:

**cpaCyGetInstances()**

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

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

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

### Context:

This function may be called from any context.

### Assumptions:

None

## 5.7 Function Documentation

### Side-Effects:

None

### Blocking:

No

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Handle to an instance of this API.
[in]	<i>errStatus</i>	The error condition that occurred
[out]	<i>pStatusText</i>	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:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed. Note, In this scenario it is INVALID to call this function a further time.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.

### Precondition:

None.

### Postcondition:

None

### Note:

None

### See also:

**CpaStatus**

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

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

This function will get the number of instances that are supported by an implementation of the Crypto 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.

## 5.7 Function Documentation

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

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

**Return values:**

*CPA\_STATUS\_SUCCESS* Function executed successfully.

*CPA\_STATUS\_FAIL* Function failed.

*CPA\_STATUS\_INVALID\_PARAM* Invalid parameter passed in.

**Precondition:**

None

**Postcondition:**

None

**Note:**

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

**See also:**

**cpaCyGetInstances**

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

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

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

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

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

This function is synchronous and blocking.

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

## 5.7 Function Documentation

[in] *numInstances* Size of the array.  
[in,out] *cyInstances* Pointer to where the instance handles will be written.

### Return values:

*CPA\_STATUS\_SUCCESS* Function executed successfully.  
*CPA\_STATUS\_FAIL* Function failed.  
*CPA\_STATUS\_INVALID\_PARAM* Invalid parameter passed in.

### Precondition:

None

### Postcondition:

None

### Note:

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

### See also:

**cpaCyGetNumInstances**

```
CpaStatus cpaCyInstanceGetInfo ( const CpaInstanceHandle instanceHandle,  
                                CpaInstanceInfo *         pInstanceInfo  
                                )
```

Function to get information on a particular instance.

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

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

No

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

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

### Return values:

*CPA\_STATUS\_SUCCESS* Function executed successfully.  
*CPA\_STATUS\_FAIL* Function failed.  
*CPA\_STATUS\_INVALID\_PARAM* Invalid parameter passed in.

## 5.7 Function Documentation

### Precondition:

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

### Postcondition:

None

### Note:

None

### See also:

**cpaCyGetNumInstances**, **cpaCyGetInstances**, **CpaInstanceInfo**

<b>CpaStatus</b>	const	
cpaCyInstanceSetNotificationCb	( <b>CpaCyInstanceNotificationCbFunc</b>	<i>pInstanceNotificationCb,</i>
	void *	<i>pCallbackTag</i>
	)	

Subscribe for instance notifications.

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

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

No

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>pInstanceNotificationCb</i>	Instance notification callback function pointer.
[in]	<i>pCallbackTag</i>	Opaque value provided by user while making individual function calls.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.

### Precondition:

Instance has been initialized.

### Postcondition:

None

## 5.7 Function Documentation

**Note:**

None

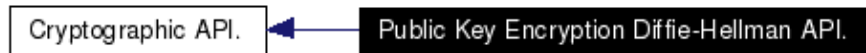
**See also:**

**CpaCylInstanceNotificationCbFunc**

# 6 Public Key Encryption Diffie-Hellman API.

[Cryptographic API.]

Collaboration diagram for Public Key Encryption Diffie-Hellman API.:



## 6.1 Detailed Description

These functions specify the API for Public Key Encryption (cryptography) operations for use with Diffie-Hellman algorithm.

## 6.2 Data Structures

- struct **\_CpaCyDhPhase1KeyGenOpData**  
Diffie-Hellman Phase 1 Key Generation Data.
- struct **\_CpaCyDhPhase2SecretKeyGenOpData**  
Diffie-Hellman Phase 2 Secret Key Generation Data.
- struct **\_CpaCyDhStats**  
Diffie-Hellman Statistics.

## 6.3 Typedefs

- typedef **\_CpaCyDhPhase1KeyGenOpData CpaCyDhPhase1KeyGenOpData**  
Diffie-Hellman Phase 1 Key Generation Data.
- typedef **\_CpaCyDhPhase2SecretKeyGenOpData CpaCyDhPhase2SecretKeyGenOpData**  
Diffie-Hellman Phase 2 Secret Key Generation Data.
- typedef **\_CpaCyDhStats CpaCyDhStats**  
Diffie-Hellman Statistics.

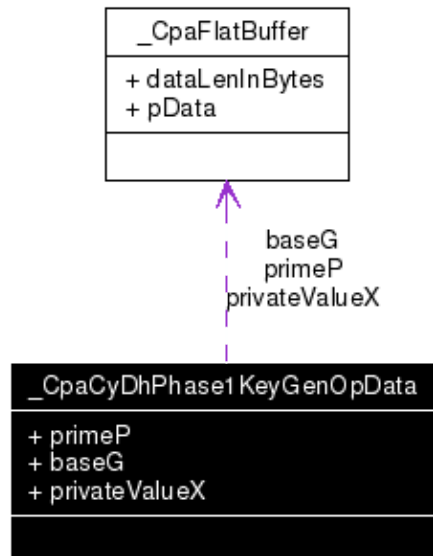
## 6.4 Functions

- **CpaStatus cpaCyDhKeyGenPhase1** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pDhPhase1Cb, void \*pCallbackTag, const **CpaCyDhPhase1KeyGenOpData** \*pPhase1KeyGenData, **CpaFlatBuffer** \*pLocalOctetStringPV)  
Function to implement Diffie-Hellman phase 1 operations.
  - **CpaStatus cpaCyDhKeyGenPhase2Secret** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pDhPhase2Cb, void \*pCallbackTag, const **CpaCyDhPhase2SecretKeyGenOpData** \*pPhase2SecretKeyGenData, **CpaFlatBuffer** \*pOctetStringSecretKey)  
Function to implement Diffie-Hellman phase 2 operations.
  - **CpaStatus cpaCyDhQueryStats** (const **CpaInstanceHandle** instanceHandle, **CpaCyDhStats** \*pDhStats)  
Query statistics for Diffie-Hellman operations.
-

## 6.5 Data Structure Documentation

### 6.5.1 \_CpaCyDhPhase1KeyGenOpData Struct Reference

Collaboration diagram for \_CpaCyDhPhase1KeyGenOpData:



#### 6.5.1.1 Detailed Description

Diffie-Hellman Phase 1 Key Generation Data.

This structure lists the different items that are required in the `cpaCyDhKeyGenPhase1` function. The client **MUST** allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the `CpaCyDhPhase1KeyGenOpData` structure.

**Note:**

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyDhKeyGenPhase1` function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. `primeP.pData[0] = MSB`.

#### 6.5.1.2 Data Fields

- **CpaFlatBuffer primeP**  
Flat buffer containing a pointer to the random odd prime number (p).
- **CpaFlatBuffer baseG**  
Flat buffer containing a pointer to base (g).
- **CpaFlatBuffer privateValueX**  
Flat buffer containing a pointer to the private value (x).

#### 6.5.1.3 Field Documentation

##### **CpaFlatBuffer \_CpaCyDhPhase1KeyGenOpData::primeP**

Flat buffer containing a pointer to the random odd prime number (p).



### 6.5.1 \_CpaCyDhPhase1KeyGenOpData Struct Reference

This number may be 768, 1024, 1536, 2048, 3072 or 4096 bits in length.

#### **CpaFlatBuffer \_CpaCyDhPhase1KeyGenOpData::baseG**

Flat buffer containing a pointer to base (g).

This MUST comply with the following:  $0 < g < p$ . The bit length of baseG MUST be less than or equal to the bit length of primeP.

#### **CpaFlatBuffer \_CpaCyDhPhase1KeyGenOpData::privateValueX**

Flat buffer containing a pointer to the private value (x).

This is a random value which MUST satisfy the following condition:  $0 < \text{PrivateValueX} < (\text{PrimeP} - 1)$

However, if a central authority specifies a private-value length L, in which case the private value (x) shall satisfy:  $2^{(L-1)} \leq \text{PrivateValueX} < 2^L$

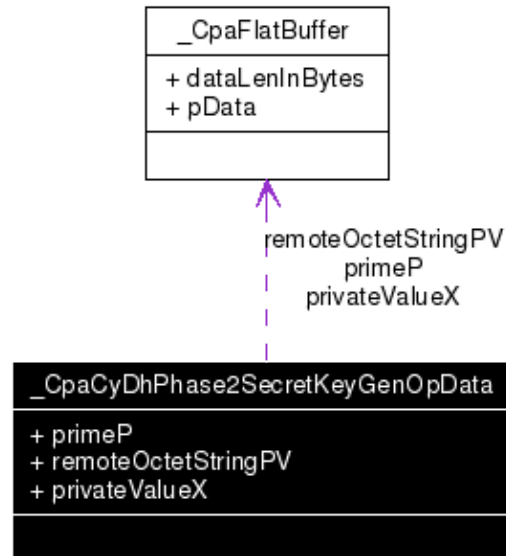
The specification defines "L" in units of bits. In this implementation only values of "L" that are multiples of 8 are permitted.

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.

---

### 6.5.2 \_CpaCyDhPhase2SecretKeyGenOpData Struct Reference

Collaboration diagram for \_CpaCyDhPhase2SecretKeyGenOpData:



#### 6.5.2.1 Detailed Description

Diffie-Hellman Phase 2 Secret Key Generation Data.

This structure lists the different items that required in the `cpaCyDhKeyGenPhase2Secret` function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

## 6.5.2 \_CpaCyDhPhase2SecretKeyGenOpData Struct Reference

### 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.

### 6.5.2.2 Data Fields

- **CpaFlatBuffer primeP**  
Flat buffer containing a pointer to the random odd prime number (p).
- **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).

### 6.5.2.3 Field Documentation

#### **CpaFlatBuffer \_CpaCyDhPhase2SecretKeyGenOpData::primeP**

Flat buffer containing a pointer to the random odd prime number (p).

This number may be 768, 1024, 1536, 2048, 3072 or 4096 bits in length. This SHOULD be same prime number as was used in the phase 1 key generation operation.

#### **CpaFlatBuffer \_CpaCyDhPhase2SecretKeyGenOpData::remoteOctetStringPV**

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

This is the public value being negotiated with. The first octet of this PV has the most significance in the integer and the last octet of PV has the least significance. The length specified MUST be equal to the length of the primeP (in bits) divided by the number of bits in an octet(8).

#### **CpaFlatBuffer \_CpaCyDhPhase2SecretKeyGenOpData::privateValueX**

Flat buffer containing a pointer to the private value (x).

This value may have been used in a call to the cpaCyDhKeyGenPhase1 function. This is a random value which MUST satisfy the following condition:  $0 < \text{privateValueX} < (\text{primeP} - 1)$ .

---

## 6.5.3 \_CpaCyDhStats Struct Reference

### 6.5.3.1 Detailed Description

Diffie-Hellman Statistics.

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

### 6.5.3.2 Data Fields

- **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**

### 6.5.3 \_CpaCyDhStats Struct Reference

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.

#### 6.5.3.3 Field Documentation

**Cpa32U \_CpaCyDhStats::numDhPhase1KeyGenRequests**

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

**Cpa32U \_CpaCyDhStats::numDhPhase1KeyGenRequestErrors**

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

**Cpa32U \_CpaCyDhStats::numDhPhase1KeyGenCompleted**

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

**Cpa32U \_CpaCyDhStats::numDhPhase1KeyGenCompletedErrors**

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

**Cpa32U \_CpaCyDhStats::numDhPhase2KeyGenRequests**

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

**Cpa32U \_CpaCyDhStats::numDhPhase2KeyGenRequestErrors**

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

**Cpa32U \_CpaCyDhStats::numDhPhase2KeyGenCompleted**

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

**Cpa32U \_CpaCyDhStats::numDhPhase2KeyGenCompletedErrors**

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

## 6.6 Typedef Documentation

```
typedef struct _CpaCyDhPhase1KeyGenOpData CpaCyDhPhase1KeyGenOpData
```

## 6.6 Typedef Documentation

Diffie-Hellman Phase 1 Key Generation Data.

This structure lists the different items that are required in the `cpaCyDhKeyGenPhase1` function. The client **MUST** allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the `CpaCyDhPhase1KeyGenOpData` structure.

**Note:**

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyDhKeyGenPhase1` function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. `primeP.pData[0] = MSB`.

```
typedef struct _CpaCyDhPhase2SecretKeyGenOpData CpaCyDhPhase2SecretKeyGenOpData
```

Diffie-Hellman Phase 2 Secret Key Generation Data.

This structure lists the different items that required in the `cpaCyDhKeyGenPhase2Secret` function. The client **MUST** allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

**Note:**

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyDhKeyGenPhase2Secret` function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. `primeP.pData[0] = MSB`.

```
typedef struct _CpaCyDhStats CpaCyDhStats
```

Diffie-Hellman Statistics.

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

---

## 6.7 Function Documentation

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

Function to implement Diffie-Hellman phase 1 operations.

This function may be used to implement the Diffie-Hellman phase 1 operations as defined in the PKCS #3 standard. It may be used to generate the two keys that are needed, the (local) octet string public value (PV) key and the private value (x) key. x is a random value, less than the prime number (PrimeP). If the length of this value is specified (non-zero) then additional constraints apply to this value. The prime number sizes specified in RFC 2409, 4306, and part of RFC 3526 are supported (bit sizes 6144 and 8192 from RFC 3536 are not supported).

**Context:**

When called as an asynchronous function it cannot sleep. It can be executed in a context that does

## 6.7 Function Documentation

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

Yes when configured to operate in synchronous mode.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pDhPhase1Cb</i>	Pointer to a callback function to be invoked when the operation is complete. If the pointer is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	Opaque User Data for this specific call. Will be returned unchanged in the callback
[in]	<i>pPhase1KeyGenData</i>	Structure containing all the data needed to perform the DH Phase 1 key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pLocalOctetStringPV</i>	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 that Diffie-Hellman is negotiating with. The first octet of this PV has the most significance in the integer and the last octet of PV has the least significance. The size of the memory required is equal to the length of pPrimeP (in bits) divided by the number of bits in an octet(8). On invocation the callback function will contain this parameter in it's pOut parameter.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized via *cpaCyStartInstance* function.

### Postcondition:

None

### Note:

When *pDhPhase1Cb* is non-NULL an asynchronous callback of type *CpaCyGenFlatBufCbFunc* is generated in response to this function call. Any errors generated during processing are reported in the structure returned in the callback.

**See also:****CpaCyGenFlatBufCbFunc, CpaCyDhPhase1KeyGenOpData**

<b>CpaStatus</b>		
cpaCyDhKeyGenPhase2Secret	( const <b>CpaInstanceHandle</b>	<i>instanceHandle,</i>
	const <b>CpaCyGenFlatBufCbFunc</b>	<i>pDhPhase2Cb,</i>
	void *	<i>pCallbackTag,</i>
	const	
	<b>CpaCyDhPhase2SecretKeyGenOpData</b>	<i>pPhase2SecretKeyGenData,</i>
	*	
	<b>CpaFlatBuffer *</b>	<i>pOctetStringSecretKey</i>
	)	

Function to implement Diffie-Hellman phase 2 operations.

This function may be used to implement the Diffie-Hellman phase 2 operation as defined in the PKCS #3 standard. It may be used to generate the Diffie-Hellman shared secret key.

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

Yes when configured to operate in synchronous mode.

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pDhPhase2Cb</i>	Pointer to a callback function to be invoked when the operation is complete. If the pointer is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in]	<i>pPhase2SecretKeyGenData</i>	Structure containing all the data needed to perform the DH Phase 2 secret key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pOctetStringSecretKey</i>	Pointer to memory allocated by the client into which the octet string secret key will be written. The size of the memory required is equal to the length of pPrimeP (in bits) divided by the number of bits in an octet(8). On invocation the callback function will contain this parameter in it's pOut parameter.

## 6.7 Function Documentation

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized via `cpaCyStartInstance` function.

### Postcondition:

None

### Note:

When `pDhPhase2Cb` is non-NULL an asynchronous callback of type `CpaCyGenFlatBufCbFunc` is generated in response to this function call. Any errors generated during processing are reported in the structure returned in the callback.

### See also:

**`CpaCyGenFlatBufCbFunc`, `CpaCyDhPhase2SecretKeyGenOpData`**

```
CpaStatus cpaCyDhQueryStats ( const CpaInstanceHandle instanceHandle,  
                             CpaCyDhStats * pDhStats  
                             )
```

Query statistics for Diffie-Hellman operations.

This function will query a specific Instance handle for Diffie- Hellman statistics. The user MUST allocate the `CpaCyDhStats` structure and pass the reference to that structure into this function call. This function writes the statistic results into the passed in `CpaCyDhStats` structure.

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

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[out]	<i>pDhStats</i>	Pointer to memory into which the statistics will be written.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
---------------------------	---------------------------------

## 6.7 Function Documentation

<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

**Precondition:**

Component has been initialized.

**Postcondition:**

None

**Note:**

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

**See also:**

**CpaCyDhStats**



# 7 Public Key Encryption DSA API.

## [Cryptographic API.]

Collaboration diagram for Public Key Encryption DSA API.:



## 7.1 Detailed Description

These functions specify the API for Public Key Encryption (cryptography) Digital Signature Algorithm (DSA) operations. The FIPS PUB 186-2 with Change Notice 1 specification is supported.

Only the modular math aspects of DSA parameter generation and message signature generation and verification are implemented here. For full DSA support, this DSA API SHOULD be used in conjunction with other parts of this overall Cryptographic API. In particular the Symmetric functions (for hashing), the Random Number Generation functions, and the Prime Number Test functions will be required.

## 7.2 Data Structures

- struct **\_CpaCyDsaPParamGenOpData**  
DSA P Parameter Generation Operation Data.
- struct **\_CpaCyDsaGParamGenOpData**  
DSA G Parameter Generation Operation Data.
- struct **\_CpaCyDsaYParamGenOpData**  
DSA Y Parameter Generation Operation Data.
- struct **\_CpaCyDsaRSignOpData**  
DSA R Sign Operation Data.
- struct **\_CpaCyDsaSSignOpData**  
DSA S Sign Operation Data.
- struct **\_CpaCyDsaRSSignOpData**  
DSA R & S Sign Operation Data.
- struct **\_CpaCyDsaVerifyOpData**  
DSA Verify Operation Data.
- struct **\_CpaCyDsaStats**  
Cryptographic DSA Statistics.

## 7.3 Typedefs

- typedef **\_CpaCyDsaPParamGenOpData CpaCyDsaPParamGenOpData**  
DSA P Parameter Generation Operation Data.
- typedef **\_CpaCyDsaGParamGenOpData CpaCyDsaGParamGenOpData**  
DSA G Parameter Generation Operation Data.
- typedef **\_CpaCyDsaYParamGenOpData CpaCyDsaYParamGenOpData**  
DSA Y Parameter Generation Operation Data.
- typedef **\_CpaCyDsaRSignOpData CpaCyDsaRSignOpData**  
DSA R Sign Operation Data.
- typedef **\_CpaCyDsaSSignOpData CpaCyDsaSSignOpData**  
DSA S Sign Operation Data.
- typedef **\_CpaCyDsaRSSignOpData CpaCyDsaRSSignOpData**  
DSA R & S Sign Operation Data.

## 7.3 Typedefs

- typedef **\_CpaCyDsaVerifyOpData CpaCyDsaVerifyOpData**  
DSA Verify Operation Data.
- typedef **\_CpaCyDsaStats CpaCyDsaStats**  
Cryptographic DSA Statistics.
- 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.
- typedef void(\* **CpaCyDsaRSSignCbFunc** )(void \*pCallbackTag, **CpaStatus** status, void \*pOpData, **CpaBoolean** protocolStatus, **CpaFlatBuffer** \*pR, **CpaFlatBuffer** \*pS)  
Definition of callback function invoked for cpaCyDsaSignRS requests.
- typedef void(\* **CpaCyDsaVerifyCbFunc** )(void \*pCallbackTag, **CpaStatus** status, void \*pOpData, **CpaBoolean** verifyStatus)  
Definition of callback function invoked for cpaCyDsaVerify requests.

## 7.4 Functions

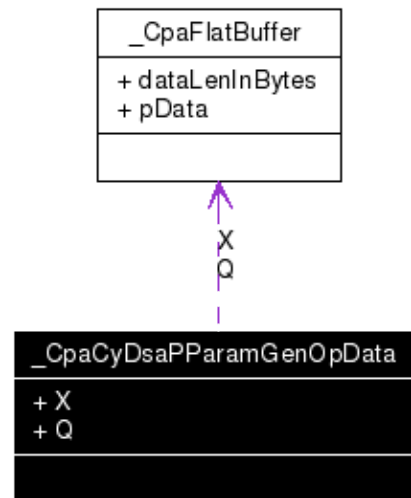
- **CpaStatus cpaCyDsaGenPParam** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaGenCbFunc** pCb, void \*pCallbackTag, const **CpaCyDsaPParamGenOpData** \*pOpData, **CpaBoolean** \*pProtocolStatus, **CpaFlatBuffer** \*pP)  
Generate DSA P Parameter.
- **CpaStatus cpaCyDsaGenGParam** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaGenCbFunc** pCb, void \*pCallbackTag, const **CpaCyDsaGParamGenOpData** \*pOpData, **CpaBoolean** \*pProtocolStatus, **CpaFlatBuffer** \*pG)  
Generate DSA G Parameter.
- **CpaStatus cpaCyDsaGenYParam** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaGenCbFunc** pCb, void \*pCallbackTag, const **CpaCyDsaYParamGenOpData** \*pOpData, **CpaBoolean** \*pProtocolStatus, **CpaFlatBuffer** \*pY)  
Generate DSA Y Parameter.
- **CpaStatus cpaCyDsaSignR** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaGenCbFunc** pCb, void \*pCallbackTag, const **CpaCyDsaRSSignOpData** \*pOpData, **CpaBoolean** \*pProtocolStatus, **CpaFlatBuffer** \*pR)  
Generate DSA R Signature.
- **CpaStatus cpaCyDsaSignS** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaGenCbFunc** pCb, void \*pCallbackTag, const **CpaCyDsaSSignOpData** \*pOpData, **CpaBoolean** \*pProtocolStatus, **CpaFlatBuffer** \*pS)  
Generate DSA S Signature.
- **CpaStatus cpaCyDsaSignRS** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaRSSignCbFunc** pCb, void \*pCallbackTag, const **CpaCyDsaRSSignOpData** \*pOpData, **CpaBoolean** \*pProtocolStatus, **CpaFlatBuffer** \*pR, **CpaFlatBuffer** \*pS)  
Generate DSA R & S Signature.
- **CpaStatus cpaCyDsaVerify** (const **CpaInstanceHandle** instanceHandle, const **CpaCyDsaVerifyCbFunc** pCb, void \*pCallbackTag, const **CpaCyDsaVerifyOpData** \*pOpData, **CpaBoolean** \*pVerifyStatus)  
Verify DSA R & S Signature.
- **CpaStatus cpaCyDsaQueryStats** (const **CpaInstanceHandle** instanceHandle, **CpaCyDsaStats** \*pDsaStats)  
Query statistics for a specific DSA instance.

---

## 7.5 Data Structure Documentation

### 7.5.1 \_CpaCyDsaPParamGenOpData Struct Reference

Collaboration diagram for \_CpaCyDsaPParamGenOpData:



#### 7.5.1.1 Detailed Description

DSA P Parameter Generation Operation Data.

This structure contains the operation data for the cpaCyDsaGenPParam function. The client **MUST** allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

**Note:**

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

**See also:**

**cpaCyDsaGenPParam()**

#### 7.5.1.2 Data Fields

- **CpaFlatBuffer X**  
 $2^{1023} \leq X < 2^{1024}$  (from FIPS 186-2 Change Notice 1 Appendix 2.2 Step 8)
- **CpaFlatBuffer Q**  
 DSA group parameter q.

#### 7.5.1.3 Field Documentation

##### CpaFlatBuffer \_CpaCyDsaPParamGenOpData::X

$2^{1023} \leq X < 2^{1024}$  (from FIPS 186-2 Change Notice 1 Appendix 2.2 Step 8)

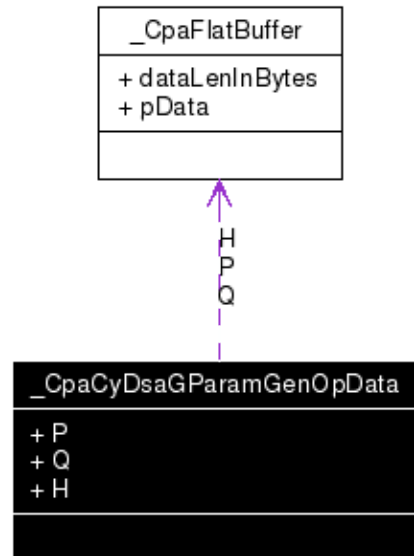
## 7.5.1 \_CpaCyDsaPParamGenOpData Struct Reference

### CpaFlatBuffer \_CpaCyDsaPParamGenOpData::Q

DSA group parameter q.

## 7.5.2 \_CpaCyDsaGParamGenOpData Struct Reference

Collaboration diagram for \_CpaCyDsaGParamGenOpData:



### 7.5.2.1 Detailed Description

DSA G Parameter Generation Operation Data.

This structure contains the operation data for the cpaCyDsaGenGParam function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

All numbers MUST be stored in big-endian order.

#### Note:

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

#### See also:

**cpaCyDsaGenGParam()**

### 7.5.2.2 Data Fields

- **CpaFlatBuffer P**  
DSA group parameter p.
- **CpaFlatBuffer Q**  
DSA group parameter q.
- **CpaFlatBuffer H**

## 7.5.2 \_CpaCyDsaGParamGenOpData Struct Reference

any integer with  $1 < h < p - 1$

### 7.5.2.3 Field Documentation

#### CpaFlatBuffer \_CpaCyDsaGParamGenOpData::P

DSA group parameter p.

#### CpaFlatBuffer \_CpaCyDsaGParamGenOpData::Q

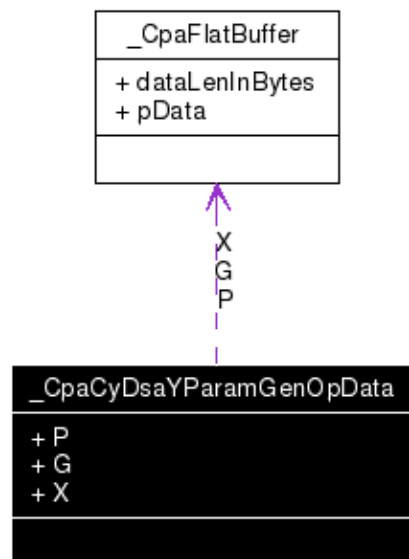
DSA group parameter q.

#### CpaFlatBuffer \_CpaCyDsaGParamGenOpData::H

any integer with  $1 < h < p - 1$

## 7.5.3 \_CpaCyDsaYParamGenOpData Struct Reference

Collaboration diagram for \_CpaCyDsaYParamGenOpData:



### 7.5.3.1 Detailed Description

DSA Y Parameter Generation Operation Data.

This structure contains the operation data for the `cpaCyDsaGenYParam` function. The client **MUST** allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

#### Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyDsaGenYParam` function, and before it has been returned in the callback, undefined behavior

### 7.5.3 \_CpaCyDsaYParamGenOpData Struct Reference

will result.

See also:

**cpaCyDsaGenYParam()**

#### 7.5.3.2 Data Fields

- **CpaFlatBuffer P**  
DSA group parameter p.
- **CpaFlatBuffer G**  
DSA group parameter g.
- **CpaFlatBuffer X**  
DSA private key x.

#### 7.5.3.3 Field Documentation

**CpaFlatBuffer \_CpaCyDsaYParamGenOpData::P**

DSA group parameter p.

**CpaFlatBuffer \_CpaCyDsaYParamGenOpData::G**

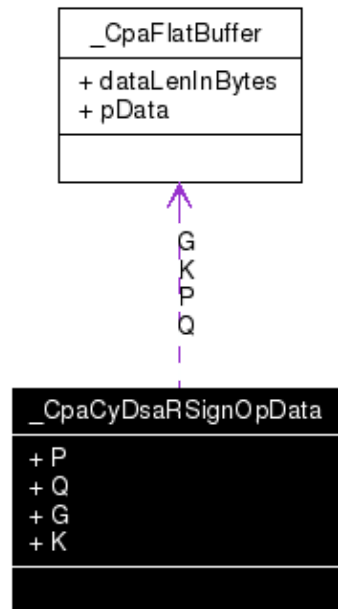
DSA group parameter g.

**CpaFlatBuffer \_CpaCyDsaYParamGenOpData::X**

DSA private key x.

### 7.5.4 \_CpaCyDsaRSignOpData Struct Reference

Collaboration diagram for \_CpaCyDsaRSignOpData:



## 7.5.4 \_CpaCyDsaRSignOpData Struct Reference

### 7.5.4.1 Detailed Description

DSA R Sign Operation Data.

This structure contains the operation data for the `cpaCyDsaSignR` function. The client **MUST** allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

#### Note:

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

#### See also:

**`cpaCyDsaSignR()`**

### 7.5.4.2 Data Fields

- **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.

### 7.5.4.3 Field Documentation

**CpaFlatBuffer \_CpaCyDsaRSignOpData::P**

DSA group parameter p.

**CpaFlatBuffer \_CpaCyDsaRSignOpData::Q**

DSA group parameter q.

**CpaFlatBuffer \_CpaCyDsaRSignOpData::G**

DSA group parameter g.

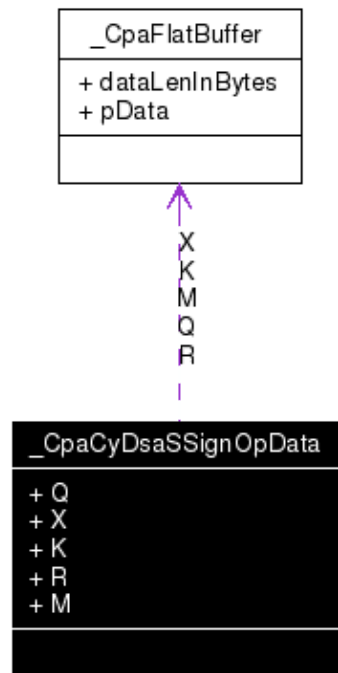
**CpaFlatBuffer \_CpaCyDsaRSignOpData::K**

DSA secret parameter k for signing.

---

### 7.5.5 \_CpaCyDsaSSignOpData Struct Reference

Collaboration diagram for \_CpaCyDsaSSignOpData:



#### 7.5.5.1 Detailed Description

DSA S Sign Operation Data.

This structure contains the operation data for the `cpaCyDsaSignS` function. The client **MUST** allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

#### Note:

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

#### See also:

**`cpaCyDsaSignS()`**

#### 7.5.5.2 Data Fields

- **CpaFlatBuffer Q**  
DSA group parameter q.
- **CpaFlatBuffer X**  
DSA private key x.
- **CpaFlatBuffer K**



7.5.5 \_CpaCyDsaSSignOpData Struct Reference

- DSA secret parameter k for signing.
- **CpaFlatBuffer R**  
DSA message signature r.
- **CpaFlatBuffer M**  
DSA message digest.

7.5.5.3 Field Documentation

**CpaFlatBuffer \_CpaCyDsaSSignOpData::Q**  
DSA group parameter q.

**CpaFlatBuffer \_CpaCyDsaSSignOpData::X**  
DSA private key x.

**CpaFlatBuffer \_CpaCyDsaSSignOpData::K**  
DSA secret parameter k for signing.

**CpaFlatBuffer \_CpaCyDsaSSignOpData::R**  
DSA message signature r.

**CpaFlatBuffer \_CpaCyDsaSSignOpData::M**  
DSA message digest.

7.5.6 \_CpaCyDsaRSSignOpData Struct Reference

Collaboration diagram for \_CpaCyDsaRSSignOpData:



## 7.5.6 \_CpaCyDsaRSSignOpData Struct Reference

### 7.5.6.1 Detailed Description

DSA R & S Sign Operation Data.

This structure contains the operation data for the `cpaCyDsaSignRS` function. The client **MUST** allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

#### Note:

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

#### See also:

**`cpaCyDsaSignRS()`**

### 7.5.6.2 Data Fields

- **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 M**  
DSA message digest.

### 7.5.6.3 Field Documentation

**CpaFlatBuffer \_CpaCyDsaRSSignOpData::P**

DSA group parameter p.

**CpaFlatBuffer \_CpaCyDsaRSSignOpData::Q**

DSA group parameter q.

**CpaFlatBuffer \_CpaCyDsaRSSignOpData::G**

DSA group parameter g.

**CpaFlatBuffer \_CpaCyDsaRSSignOpData::X**

DSA private key x.

**CpaFlatBuffer \_CpaCyDsaRSSignOpData::K**

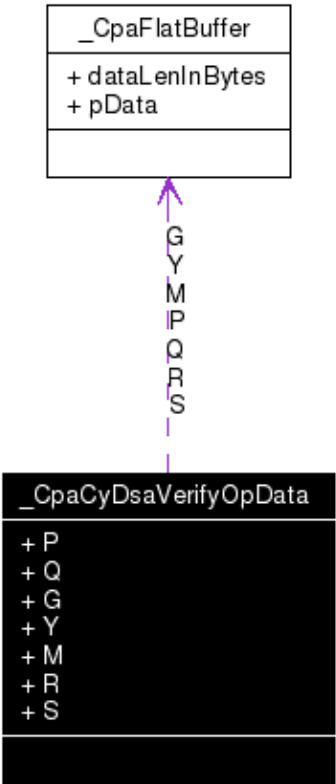
DSA secret parameter k for signing.

**CpaFlatBuffer \_CpaCyDsaRSSignOpData::M**

DSA message digest.

7.5.7 \_CpaCyDsaVerifyOpData Struct Reference

Collaboration diagram for \_CpaCyDsaVerifyOpData:



7.5.7.1 Detailed Description

DSA Verify Operation Data.

This structure contains the operation data for the cpaCyDsaVerify function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

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

**See also:**  
cpaCyDsaVerify()

## 7.5.7 \_CpaCyDsaVerifyOpData Struct Reference

### 7.5.7.2 Data Fields

- **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 M**  
DSA message digest.
- **CpaFlatBuffer R**  
DSA message signature r.
- **CpaFlatBuffer S**  
DSA message signature s.

### 7.5.7.3 Field Documentation

#### **CpaFlatBuffer \_CpaCyDsaVerifyOpData::P**

DSA group parameter p.

#### **CpaFlatBuffer \_CpaCyDsaVerifyOpData::Q**

DSA group parameter q.

#### **CpaFlatBuffer \_CpaCyDsaVerifyOpData::G**

DSA group parameter g.

#### **CpaFlatBuffer \_CpaCyDsaVerifyOpData::Y**

DSA public key y.

#### **CpaFlatBuffer \_CpaCyDsaVerifyOpData::M**

DSA message digest.

#### **CpaFlatBuffer \_CpaCyDsaVerifyOpData::R**

DSA message signature r.

#### **CpaFlatBuffer \_CpaCyDsaVerifyOpData::S**

DSA message signature s.

---

## 7.5.8 \_CpaCyDsaStats Struct Reference

### 7.5.8.1 Detailed Description

Cryptographic DSA Statistics.

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

## 7.5.8.2 Data Fields

- **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.

## 7.5.8 \_CpaCyDsaStats Struct Reference

- **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.

### 7.5.8.3 Field Documentation

#### **Cpa32U \_CpaCyDsaStats::numDsaPParamGenRequests**

Total number of successful DSA P parameter generation requests.

#### **Cpa32U \_CpaCyDsaStats::numDsaPParamGenRequestErrors**

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

#### **Cpa32U \_CpaCyDsaStats::numDsaPParamGenCompleted**

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

#### **Cpa32U \_CpaCyDsaStats::numDsaPParamGenCompletedErrors**

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

#### **Cpa32U \_CpaCyDsaStats::numDsaGParamGenRequests**

Total number of successful DSA G parameter generation requests.

#### **Cpa32U \_CpaCyDsaStats::numDsaGParamGenRequestErrors**

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

#### **Cpa32U \_CpaCyDsaStats::numDsaGParamGenCompleted**

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

#### **Cpa32U \_CpaCyDsaStats::numDsaGParamGenCompletedErrors**

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

#### **Cpa32U \_CpaCyDsaStats::numDsaYParamGenRequests**

Total number of successful DSA Y parameter generation requests.

#### **Cpa32U \_CpaCyDsaStats::numDsaYParamGenRequestErrors**

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

#### **Cpa32U \_CpaCyDsaStats::numDsaYParamGenCompleted**

## 7.5.8 \_CpaCyDsaStats Struct Reference

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

### **Cpa32U \_CpaCyDsaStats::numDsaYParamGenCompletedErrors**

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

### **Cpa32U \_CpaCyDsaStats::numDsaRSignRequests**

Total number of successful DSA R sign generation requests.

### **Cpa32U \_CpaCyDsaStats::numDsaRSignRequestErrors**

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

### **Cpa32U \_CpaCyDsaStats::numDsaRSignCompleted**

Total number of DSA R sign operations that completed successfully.

### **Cpa32U \_CpaCyDsaStats::numDsaRSignCompletedErrors**

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

### **Cpa32U \_CpaCyDsaStats::numDsaSSignRequests**

Total number of successful DSA S sign generation requests.

### **Cpa32U \_CpaCyDsaStats::numDsaSSignRequestErrors**

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

### **Cpa32U \_CpaCyDsaStats::numDsaSSignCompleted**

Total number of DSA S sign operations that completed successfully.

### **Cpa32U \_CpaCyDsaStats::numDsaSSignCompletedErrors**

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

### **Cpa32U \_CpaCyDsaStats::numDsaRSSignRequests**

Total number of successful DSA RS sign generation requests.

### **Cpa32U \_CpaCyDsaStats::numDsaRSSignRequestErrors**

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

### **Cpa32U \_CpaCyDsaStats::numDsaRSSignCompleted**

Total number of DSA RS sign operations that completed successfully.

### **Cpa32U \_CpaCyDsaStats::numDsaRSSignCompletedErrors**

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

### **Cpa32U \_CpaCyDsaStats::numDsaVerifyRequests**

Total number of successful DSA verify generation requests.

### **Cpa32U \_CpaCyDsaStats::numDsaVerifyRequestErrors**

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

### **Cpa32U \_CpaCyDsaStats::numDsaVerifyCompleted**

Total number of DSA verify operations that completed successfully.

### **Cpa32U \_CpaCyDsaStats::numDsaVerifyCompletedErrors**

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

### **Cpa32U \_CpaCyDsaStats::numDsaVerifyFailures**

Total number of DSA verify operations that executed successfully but the outcome of the test was that the verification failed.

N.B. This does not indicate an "error".

---

## 7.6 Typedef Documentation

### **typedef struct \_CpaCyDsaPParamGenOpData CpaCyDsaPParamGenOpData**

DSA P Parameter Generation Operation Data.

This structure contains the operation data for the cpaCyDsaGenPParam function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

#### **Note:**

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

#### **See also:**

**cpaCyDsaGenPParam()**

### **typedef struct \_CpaCyDsaGParamGenOpData CpaCyDsaGParamGenOpData**

DSA G Parameter Generation Operation Data.

This structure contains the operation data for the cpaCyDsaGenGParam function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

All numbers MUST be stored in big-endian order.

#### **Note:**

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

#### **See also:**

**cpaCyDsaGenGParam()**

### **typedef struct \_CpaCyDsaYParamGenOpData CpaCyDsaYParamGenOpData**

DSA Y Parameter Generation Operation Data.



## 7.6 Typedef Documentation

This structure contains the operation data for the `cpaCyDsaGenYParam` function. The client **MUST** allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

**Note:**

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

**See also:**

**`cpaCyDsaGenYParam()`**

```
typedef struct _CpaCyDsaRSignOpData CpaCyDsaRSignOpData
```

DSA R Sign Operation Data.

This structure contains the operation data for the `cpaCyDsaSignR` function. The client **MUST** allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

**Note:**

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

**See also:**

**`cpaCyDsaSignR()`**

```
typedef struct _CpaCyDsaSSignOpData CpaCyDsaSSignOpData
```

DSA S Sign Operation Data.

This structure contains the operation data for the `cpaCyDsaSignS` function. The client **MUST** allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

**Note:**

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

**See also:**

### **cpaCyDsaSignS()**

```
typedef struct _CpaCyDsaRSSignOpData CpaCyDsaRSSignOpData  
DSA R & S Sign Operation Data.
```

This structure contains the operation data for the `cpaCyDsaSignRS` function. The client **MUST** allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

#### **Note:**

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

#### **See also:**

**cpaCyDsaSignRS()**

```
typedef struct _CpaCyDsaVerifyOpData CpaCyDsaVerifyOpData  
DSA Verify Operation Data.
```

This structure contains the operation data for the `cpaCyDsaVerify` function. The client **MUST** allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

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

#### **Note:**

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

#### **See also:**

**cpaCyDsaVerify()**

```
typedef struct _CpaCyDsaStats CpaCyDsaStats  
Cryptographic DSA Statistics.
```

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

```
typedef 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.

This is the prototype for the `cpaCyDsaGenCbFunc` callback function.

#### **Context:**

## 7.6 Typedef Documentation

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

### Assumptions:

None

### Side-Effects:

None

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>pCallbackTag</i>	User-supplied value to help identify request.
[in]	<i>status</i>	Status of the operation. Valid values are CPA_STATUS_SUCCESS and CPA_STATUS_FAIL.
[in]	<i>pOpData</i>	Opaque pointer to Operation data supplied in request.
[in]	<i>protocolStatus</i>	The result passes/fails the DSA protocol related checks.
[in]	<i>pOut</i>	Output data from the request.

### Return values:

None

### Precondition:

Component has been initialized.

### Postcondition:

None

### Note:

None

### See also:

**cpaCyDsaGenPParam()** **cpaCyDsaGenGParam()** **cpaCyDsaSignR()** **cpaCyDsaSignS()**

```
typedef void(* CpaCyDsaRSSignCbFunc)(void *pCallbackTag, CpaStatus status, void *pOpData,  
CpaBoolean protocolStatus, CpaFlatBuffer *pR, CpaFlatBuffer *pS)
```

Definition of callback function invoked for cpaCyDsaSignRS requests.

This is the prototype for the cpaCyDsaSignRS callback function, which will provide the DSA message signature r and s parameters.

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Reentrant:

No

## 7.6 Typedef Documentation

### Thread-safe:

Yes

### Parameters:

[in]	<i>pCallbackTag</i>	User-supplied value to help identify request.
[in]	<i>status</i>	Status of the operation. Valid values are CPA_STATUS_SUCCESS and CPA_STATUS_FAIL.
[in]	<i>pOpData</i>	Operation data pointer supplied in request.
[in]	<i>protocolStatus</i>	The result passes/fails the DSA protocol related checks.
[in]	<i>pR</i>	DSA message signature r.
[in]	<i>pS</i>	DSA message signature s.

### Return values:

None

### Precondition:

Component has been initialized.

### Postcondition:

None

### Note:

None

### See also:

**cpaCyDsaSignRS()**

```
typedef void(* CpaCyDsaVerifyCbFunc)(void *pCallbackTag, CpaStatus status, void *pOpData, CpaBoolean verifyStatus)
```

Definition of callback function invoked for cpaCyDsaVerify requests.

This is the prototype for the cpaCyDsaVerify callback function.

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>pCallbackTag</i>	User-supplied value to help identify request.
[in]	<i>status</i>	Status of the operation. Valid values are CPA_STATUS_SUCCESS and CPA_STATUS_FAIL.
[in]	<i>pOpData</i>	Operation data pointer supplied in request.
[in]	<i>verifyStatus</i>	The verification passed or failed.

## 7.7 Function Documentation

**Return values:**

None

**Precondition:**

Component has been initialized.

**Postcondition:**

None

**Note:**

None

**See also:**

**cpaCyDsaVerify()**

---

## 7.7 Function Documentation

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

Generate DSA P Parameter.

This function performs FIPS 186-2 Change Notice 1 Appendix 2.2 Step 9 and part of Step 11:

Step 9. Let  $c = X \bmod 2q$  and set  $p = X - (c - 1)$ . Step 11. Perform a robust primality test on  $p$ . [a GCD test against ~1400 small primes is performed on  $p$  to eliminate ~94% of composites - this is NOT a "robust" primality test]

A response status of ok (`protocolStatus == CPA_TRUE`) means  $p$  is in the right range, and SHOULD be subjected to a robust primality test (for example 50 rounds of Miller-Rabin).

A response status of not ok (`protocolStatus == CPA_FALSE`) means  $p$  is either in the right range but composite, or  $p < 2^{1023}$  (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:**

## 7.7 Function Documentation

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pCb</i>	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	User-supplied value to help identify request.
[in]	<i>pOpData</i>	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pProtocolStatus</i>	The result passes/fails the DSA protocol related checks.
[out]	<i>pP</i>	Candidate for DSA parameter p, p odd and $2^{1023} < p < X$ On invocation the callback function will contain this parameter in it's pOut parameter.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized.

### Postcondition:

None

### Note:

When pCb is non-NULL an asynchronous callback of type CpaCyDsaPParamGenCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

### See also:

**CpaCyDsaPParamGenOpData, CpaCyDsaGenCbFunc**

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

Generate DSA G Parameter.

This function performs FIPS 186-2 Change Notice 1 Appendix 4 Step 2, Step 4, and part of Step 5:

Step 2. Let  $e = (p - 1)/q$ . Step 4. Set  $g = h^e \bmod p$ . Step 5. If  $g = 1$ , go to step 3. [a check for  $g = 1$  is performed, and status returned accordingly]

A response status of ok (`protocolStatus == CPA_TRUE`) means g is acceptable.

## 7.7 Function Documentation

A response status of not ok (`protocolStatus == CPA_FALSE`) means  $g = 1$ , so a different value of  $h$  SHOULD be used to generate another value of  $g$ .

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

Yes when configured to operate in synchronous mode.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pCb</i>	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	User-supplied value to help identify request.
[in]	<i>pOpData</i>	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pProtocolStatus</i>	The result passes/fails the DSA protocol related checks.
[out]	<i>pG</i>	$g = h^((p-1)/q) \bmod p$ . On invocation the callback function will contain this parameter in it's <code>pOut</code> parameter.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized via `cpaCyStartInstance` function.

### Postcondition:

None

### Note:

When `pCb` is non-NULL an asynchronous callback of type `CpaCyDsaGParamGenCbFunc` is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

### See also:

**CpaCyDsaGParamGenOpData, CpaCyDsaGenCbFunc**

## 7.7 Function Documentation

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

Generate DSA Y Parameter.

This function performs FIPS 186-2 Change Notice 1 Section 4 Step 5:  $y = g^x \text{ mod } p$

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

Yes when configured to operate in synchronous mode.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pCb</i>	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	User-supplied value to help identify request.
[in]	<i>pOpData</i>	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pProtocolStatus</i>	The result passes/fails the DSA protocol related checks.
[out]	<i>pY</i>	$y = g^x \text{ mod } p$ * On invocation the callback function will contain this parameter in it's pOut parameter.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized via `cpaCyStartInstance` function.

### Postcondition:



## 7.7 Function Documentation

None

### Note:

When pCb is non-NULL an asynchronous callback of type CpaCyDsaYParamGenCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

### See also:

**CpaCyDsaYParamGenOpData, CpaCyDsaGenCbFunc**

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

Generate DSA R Signature.

This function performs FIPS 186-2 Change Notice 1 Section 5:  $r = (g^k \bmod p) \bmod q$

A response status of ok (protocolStatus == CPA\_TRUE) means  $r \neq 0$ . A response status of not ok (protocolStatus == CPA\_FALSE) means  $r = 0$ .

Generation of signature r does not depend on the content of the message being signed, so this operation can be done in advance for different values of k. Then once each message becomes available only the signature s needs to be generated.

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

Yes when configured to operate in synchronous mode.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pCb</i>	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	User-supplied value to help identify request.
[in]	<i>pOpData</i>	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes

## 7.7 Function Documentation

ownership of the memory until it is returned in the callback.

[out] *pProtocolStatus* The result passes/fails the DSA protocol related checks.

[out] *pR* DSA message signature r. On invocation the callback function will contain this parameter in it's 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.

### Precondition:

The component has been initialized via *cpaCyStartInstance* function.

### Postcondition:

None

### Note:

When *pCb* is non-NULL an asynchronous callback of type *CpaCyDsaRSignCbFunc* is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

### See also:

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

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

Generate DSA S Signature.

This function performs FIPS 186-2 Change Notice 1 Section 5:  $s = (k^{-1}(\text{SHA-1}(M) + xr)) \bmod q$

This function does not perform the SHA-1 digest, instead the caller MUST provide the message digest in the request.

A response status of ok (*protocolStatus* == *CPA\_TRUE*) means  $s \neq 0$ . A response status of not ok (*protocolStatus* == *CPA\_FALSE*) means  $s = 0$ .

If signature *r* has been generated in advance, then this function can be used to generate the signature *s* once the message becomes available.

### Context:

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

### Assumptions:

None

## 7.7 Function Documentation

### Side-Effects:

None

### Blocking:

Yes when configured to operate in synchronous mode.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pCb</i>	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	User-supplied value to help identify request.
[in]	<i>pOpData</i>	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pProtocolStatus</i>	The result passes/fails the DSA protocol related checks.
[out]	<i>pS</i>	DSA message signature s. On invocation the callback function will contain this parameter in it's pOut parameter.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized via `cpaCyStartInstance` function.

### Postcondition:

None

### Note:

When *pCb* is non-NULL an asynchronous callback of type `CpaCyDsaSSignCbFunc` is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

### See also:

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

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

## 7.7 Function Documentation

Generate DSA R & S Signature.

This function performs FIPS 186-2 Change Notice 1 Section 5:

$r = (g^k \bmod p) \bmod q$   $s = (k^{-1}(\text{SHA-1}(M) + xr)) \bmod q$  [this function does not perform the SHA-1 digest, instead the caller MUST provide the message digest in the request]

A response status of ok (`protocolStatus == CPA_TRUE`) means  $r \neq 0$  and  $s \neq 0$ .

A response status of not ok (`protocolStatus == CPA_FALSE`) means  $r = 0$  or  $s = 0$ .

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

Yes when configured to operate in synchronous mode.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pCb</i>	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	User-supplied value to help identify request.
[in]	<i>pOpData</i>	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pProtocolStatus</i>	The result passes/fails the DSA protocol related checks.
[out]	<i>pR</i>	DSA message signature r.
[out]	<i>pS</i>	DSA message signature s.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized via `cpaCyStartInstance` function.

### Postcondition:

None

## 7.7 Function Documentation

### Note:

When pCb is non-NULL an asynchronous callback of type CpaCyDsaRSSignCbFunc is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

### See also:

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

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

Verify DSA R & S Signature.

This function performs FIPS 186-2 Change Notice 1 Section 6:  $w = (s')^{-1} \bmod q$   $u1 = ((SHA-1(M'))w) \bmod q$   $u2 = ((r')w) \bmod q$   $v = (((g)^{u1} (y)^{u2}) \bmod p) \bmod q$

A response status of ok (verifyStatus == CPA\_TRUE) means  $v = r'$ . A response status of not ok (verifyStatus == CPA\_FALSE) means  $v \neq r'$ .

This function does not perform the SHA-1 digest, instead the caller MUST provide the message digest in the request.

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

Yes when configured to operate in synchronous mode.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pCb</i>	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	User-supplied value to help identify request.
[in]	<i>pOpData</i>	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pVerifyStatus</i>	The verification passed or failed.

## 7.7 Function Documentation

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized via `cpaCyStartInstance` function.

### Postcondition:

None

### Note:

When `pCb` is non-NULL an asynchronous callback of type `CpaCyDsaVerifyCbFunc` is generated in response to this function call. For optimal performance, data pointers SHOULD be 8-byte aligned.

### See also:

**CpaCyDsaVerifyOpData, CpaCyDsaVerifyCbFunc**

```
CpaStatus cpaCyDsaQueryStats ( const CpaInstanceHandle instanceHandle,  
                               CpaCyDsaStats * pDsaStats  
                               )
```

Query statistics for a specific DSA instance.

This function will query a specific instance of the DSA implementation for statistics. The user MUST allocate the `CpaCyDsaStats` structure and pass the reference to that structure into this function call. This function writes the statistic results into the passed in `CpaCyDsaStats` structure.

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

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

This function is synchronous and blocking.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[out]	<i>pDsaStats</i>	Pointer to memory into which the statistics will be written.

## 7.7 Function Documentation

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

Component has been initialized.

### Postcondition:

None

### Note:

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

### See also:

**CpaCyDsaStats**

# 8 Crypto Instance Maintenance API.

## [Cryptographic API.]

Collaboration diagram for Crypto Instance Maintenance API.:



## 8.1 Detailed Description

These functions specify the Instance Maintenance API for available Crypto Instances.

It's expected that these functions will only be called via a single system maintenance entity, rather than individual clients

## 8.2 Functions

- **CpaStatus cpaCyStartInstance (CpaInstanceHandle instanceHandle)**  
Cryptographic Component Initialization and Start function.
  - **CpaStatus cpaCyStopInstance (CpaInstanceHandle instanceHandle)**  
Cryptographic Component Stop function.
- 

## 8.3 Function Documentation

**CpaStatus cpaCyStartInstance ( CpaInstanceHandle instanceHandle )**

Cryptographic Component Initialization and Start function.

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

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

This function is synchronous and blocking.

**Reentrant:**

No

**Thread-safe:**

Yes



## 8.3 Function Documentation

### Parameters:

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

### Return values:

*CPA\_STATUS\_SUCCESS* Function executed successfully.

*CPA\_STATUS\_FAIL* Function failed. Suggested course of action is to shutdown and restart.

### Precondition:

None.

### Postcondition:

None

### Note:

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

### See also:

**cpaCyStopInstance()**

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

Cryptographic Component Stop function.

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

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

This function is synchronous and blocking.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

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

### Return values:

*CPA\_STATUS\_SUCCESS* Function executed successfully.

*CPA\_STATUS\_FAIL* Function failed. Suggested course of action is to ensure requests are not still being submitted and that all sessions are deregistered. If this does not help, then forcefully remove the component from the system.

### 8.3 Function Documentation

**Precondition:**

The component has been initialized via `cpaCyStartInstance`

**Postcondition:**

None

**Note:**

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

**See also:**

`cpaCyStartInstance()`

# 9 Key and Mask Generation API.

## [Cryptographic API.]

Collaboration diagram for Key and Mask Generation API.:



## 9.1 Detailed Description

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

## 9.2 Data Structures

- struct **\_CpaCyKeyGenSslOpData**  
SSL data for key generation functions.
- struct **\_CpaCyKeyGenTlsOpData**  
TLS data for key generation functions.
- struct **\_CpaCyKeyGenMgfOpData**  
Key Generation Mask Generation Function (MGF) Data.
- struct **\_CpaCyKeyGenStats**  
Key Generation Statistics.

## 9.3 Defines

- #define **CPA\_CY\_KEY\_GEN\_SSL\_TLS\_RANDOM\_LEN\_IN\_BYTES**  
SSL or TLS key generation random number length.
- #define **CPA\_CY\_KEY\_GEN\_SSL\_TLS\_SEED\_LEN\_IN\_BYTES**  
TLS seed length.

## 9.4 Typedefs

- typedef enum **\_CpaCyKeySslOp** **CpaCyKeySslOp**  
SSL Operation Types.
- typedef **\_CpaCyKeyGenSslOpData** **CpaCyKeyGenSslOpData**  
SSL data for key generation functions.
- typedef enum **\_CpaCyKeyTlsOp** **CpaCyKeyTlsOp**  
TLS Operation Types.
- typedef **\_CpaCyKeyGenTlsOpData** **CpaCyKeyGenTlsOpData**  
TLS data for key generation functions.
- typedef **\_CpaCyKeyGenMgfOpData** **CpaCyKeyGenMgfOpData**  
Key Generation Mask Generation Function (MGF) Data.
- typedef **\_CpaCyKeyGenStats** **CpaCyKeyGenStats**  
Key Generation Statistics.

## 9.5 Enumerations

- enum **\_CpaCyKeySslOp** {  
    **CPA\_CY\_KEY\_SSL\_OP\_MASTER\_SECRET\_DERIVE,**  
    **CPA\_CY\_KEY\_SSL\_OP\_KEY\_MATERIAL\_DERIVE,**

## 9.5 Enumerations

```
CPA_CY_KEY_SSL_OP_USER_DEFINED
}
    SSL Operation Types.
• enum _CpaCyKeyTlsOp {
    CPA_CY_KEY_TLS_OP_MASTER_SECRET_DERIVE,
    CPA_CY_KEY_TLS_OP_KEY_MATERIAL_DERIVE,
    CPA_CY_KEY_TLS_OP_CLIENT_FINISHED_DERIVE,
    CPA_CY_KEY_TLS_OP_SERVER_FINISHED_DERIVE,
    CPA_CY_KEY_TLS_OP_USER_DEFINED
}
    TLS Operation Types.
```

## 9.6 Functions

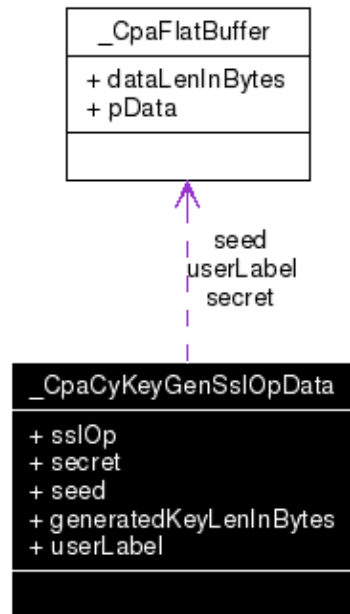
- **CpaStatus cpaCyKeyGenSsl** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pKeyGenCb, void \*pCallbackTag, const **CpaCyKeyGenSslOpData** \*pKeyGenSslOpData, **CpaFlatBuffer** \*pGeneratedKeyBuffer)  
SSL Key Generation Function.
  - **CpaStatus cpaCyKeyGenTls** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pKeyGenCb, void \*pCallbackTag, const **CpaCyKeyGenTlsOpData** \*pKeyGenTlsOpData, **CpaFlatBuffer** \*pGeneratedKeyBuffer)  
TLS Key Generation Function.
  - **CpaStatus cpaCyKeyGenMgf** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pKeyGenCb, void \*pCallbackTag, const **CpaCyKeyGenMgfOpData** \*pKeyGenMgfOpData, **CpaFlatBuffer** \*pGeneratedMaskBuffer)  
Mask Generation Function.
  - **CpaStatus cpaCyKeyGenQueryStats** (const **CpaInstanceHandle** instanceHandle, **CpaCyKeyGenStats** \*pKeyGenStats)  
Key and Mask generation statistics specific to an instance.
- 

## 9.7 Data Structure Documentation

### 9.7.1 \_CpaCyKeyGenSslOpData Struct Reference

Collaboration diagram for \_CpaCyKeyGenSslOpData:

### 9.7.1 \_CpaCyKeyGenSslOpData Struct Reference



#### 9.7.1.1 Detailed Description

SSL data for key generation functions.

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

SSL Master-Secret Derivation: `sslOp` = `CPA_CY_KEY_SSL_OP_MASTER_SECRET_DERIVE` `secret` = pre-master secret key `seed` = `client_random` + `server_random` `userLabel` = NULL

SSL Key-Material Derivation: `sslOp` = `CPA_CY_KEY_SSL_OP_KEY_MATERIAL_DERIVE` `secret` = master secret key `seed` = `server_random` + `client_random` `userLabel` = NULL

(note that the client/server random order is reversed from that used for Master-Secret Derivation)

Notes: 1. Each of the client and server random numbers need to be of length `CPA_CY_KEY_GEN_SSL_TLS_RANDOM_LEN_IN_BYTES`. 2. In each of the above descriptions, + indicates concatenation. 3. 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`.

#### 9.7.1.2 Data Fields

- **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.
- **CpaFlatBuffer seed**  
Flat buffer containing a pointer to the seed data.
- **Cpa32U generatedKeyLenInBytes**  
The requested length of the generated key in bytes.
- **CpaFlatBuffer userLabel**  
Optional flat buffer containing a pointer to a user defined label.

## 9.7.1 \_CpaCyKeyGenSslOpData Struct Reference

### 9.7.1.3 Field Documentation

#### CpaCyKeySslOp \_CpaCyKeyGenSslOpData::sslOp

Indicate the SSL operation to be performed.

#### CpaFlatBuffer \_CpaCyKeyGenSslOpData::secret

Flat buffer containing a pointer to either the master or pre-master secret key.

The length field indicates the length of the secret key in bytes. Implementation-specific limits may apply to this length.

#### CpaFlatBuffer \_CpaCyKeyGenSslOpData::seed

Flat buffer containing a pointer to the seed data.

The length field needs to be equal to CPA\_CY\_KEY\_GEN\_SSL\_TLS\_SEED\_LEN\_IN\_BYTES.

#### Cpa32U \_CpaCyKeyGenSslOpData::generatedKeyLenInBytes

The requested length of the generated key in bytes.

Implementation-specific limits may apply to this length.

#### CpaFlatBuffer \_CpaCyKeyGenSslOpData::userLabel

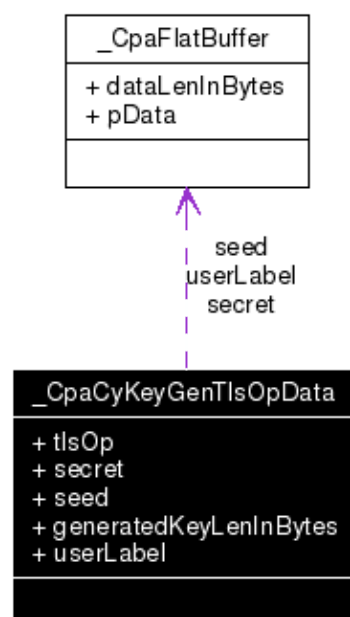
Optional flat buffer containing a pointer to a user defined label.

The length field indicates the length of the label in bytes. To use this field, the sslOp must be CPA\_CY\_KEY\_SSL\_OP\_USER\_DEFINED, otherwise it is ignored and can be set to NULL. Implementation-specific limits may apply to this length.

---

## 9.7.2 \_CpaCyKeyGenTlsOpData Struct Reference

Collaboration diagram for \_CpaCyKeyGenTlsOpData:



## 9.7.2 \_CpaCyKeyGenTlsOpData Struct Reference

### 9.7.2.1 Detailed Description

TLS data for key generation functions.

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

TLS Master-Secret Derivation: `tlsOp = CPA_CY_KEY_TLS_OP_MASTER_SECRET_DERIVE` `secret = pre-master secret key seed = client_random + server_random` `userLabel = NULL`

TLS Key-Material Derivation: `tlsOp = CPA_CY_KEY_TLS_OP_KEY_MATERIAL_DERIVE` `secret = master secret key seed = server_random + client_random` `userLabel = NULL`

(note that the client/server random order is reversed from that used for Master-Secret Derivation)

TLS Client finished/Server finished tag Derivation: `tlsOp = CPA_CY_KEY_TLS_OP_CLIENT_FINISHED_DERIVE` (Client) or `CPA_CY_KEY_TLS_OP_SERVER_FINISHED_DERIVE` (server) `secret = master secret key seed = MD5(handshake_messages) + SHA-1(handshake_messages)` `userLabel = NULL`

Notes: 1. Each of the client and server random seeds need to be of length `CPA_CY_KEY_GEN_SSL_TLS_RANDOM_LEN_IN_BYTES`. 2. In each of the above descriptions, + indicates concatenation. 3. The label used is predetermined by the TLS operation in line with the TLS 1.0 specification, and can be overridden by using a user defined operation `CPA_CY_KEY_TLS_OP_USER_DEFINED` and associated `userLabel`.

### 9.7.2.2 Data Fields

- **CpaCyKeyTlsOp `tlsOp`**
- **CpaFlatBuffer `secret`**  
Flat buffer containing a pointer to either the master or pre-master secret key.
- **CpaFlatBuffer `seed`**  
Flat buffer containing a pointer to the seed data.
- **Cpa32U `generatedKeyLenInBytes`**  
The requested length of the generated key in bytes.
- **CpaFlatBuffer `userLabel`**  
Optional flat buffer containing a pointer to a user defined label.

### 9.7.2.3 Field Documentation

#### **CpaFlatBuffer \_CpaCyKeyGenTlsOpData::secret**

Flat buffer containing a pointer to either the master or pre-master secret key.

The length field indicates the length of the secret in bytes. Implementation-specific limits may apply to this length.

#### **CpaFlatBuffer \_CpaCyKeyGenTlsOpData::seed**

Flat buffer containing a pointer to the seed data.

The length field needs to be equal to `CPA_CY_KEY_GEN_SSL_TLS_SEED_LEN_IN_BYTES`.

#### **Cpa32U \_CpaCyKeyGenTlsOpData::generatedKeyLenInBytes**

The requested length of the generated key in bytes.

### 9.7.3 \_CpaCyKeyGenMgfOpData Struct Reference

Implementation-specific limits may apply to this length.

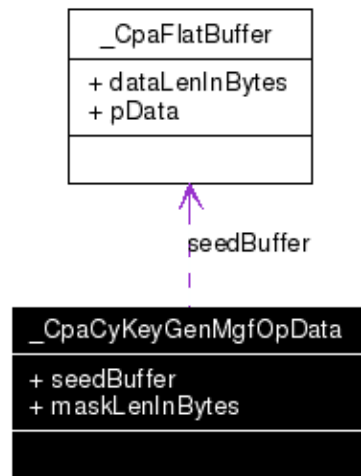
#### CpaFlatBuffer \_CpaCyKeyGenTlsOpData::userLabel

Optional flat buffer containing a pointer to a user defined label.

The length field indicates the length of the label in bytes. To use this field, the tlsOp must be CPA\_CY\_KEY\_TLS\_OP\_USER\_DEFINED, otherwise it is ignored and can be set to NULL. Implementation-specific limits may apply to this length.

### 9.7.3 \_CpaCyKeyGenMgfOpData Struct Reference

Collaboration diagram for \_CpaCyKeyGenMgfOpData:



#### 9.7.3.1 Detailed Description

Key Generation Mask Generation Function (MGF) Data.

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

#### 9.7.3.2 Data Fields

- **CpaFlatBuffer seedBuffer**  
Caller MUST allocate a buffer and populate with the input seed data.
- **Cpa32U maskLenInBytes**  
The requested length of the generated mask in bytes.

#### 9.7.3.3 Field Documentation

#### CpaFlatBuffer \_CpaCyKeyGenMgfOpData::seedBuffer

Caller MUST allocate a buffer and populate with the input seed data.

For optimal performance the start of the seed SHOULD be allocated on an 8-byte boundary. The length field represents the seed length in bytes. Implementation-specific limits may apply to this length.

#### Cpa32U \_CpaCyKeyGenMgfOpData::maskLenInBytes

The requested length of the generated mask in bytes.



Implementation-specific limits may apply to this length.

---

### 9.7.4 \_CpaCyKeyGenStats Struct Reference

#### 9.7.4.1 Detailed Description

Key Generation Statistics.

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.

#### 9.7.4.2 Data Fields

- **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.
- **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.

#### 9.7.4.3 Field Documentation

##### **Cpa32U \_CpaCyKeyGenStats::numSslKeyGenRequests**

Total number of successful SSL key generation requests.

##### **Cpa32U \_CpaCyKeyGenStats::numSslKeyGenRequestErrors**

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

##### **Cpa32U \_CpaCyKeyGenStats::numSslKeyGenCompleted**

Total number of SSL key generation operations that completed successfully.

**Cpa32U \_CpaCyKeyGenStats::numSslKeyGenCompletedErrors**

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

**Cpa32U \_CpaCyKeyGenStats::numTlsKeyGenRequests**

Total number of successful TLS key generation requests.

**Cpa32U \_CpaCyKeyGenStats::numTlsKeyGenRequestErrors**

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

**Cpa32U \_CpaCyKeyGenStats::numTlsKeyGenCompleted**

Total number of TLS key generation operations that completed successfully.

**Cpa32U \_CpaCyKeyGenStats::numTlsKeyGenCompletedErrors**

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

**Cpa32U \_CpaCyKeyGenStats::numMgfKeyGenRequests**

Total number of successful MGF key generation requests.

**Cpa32U \_CpaCyKeyGenStats::numMgfKeyGenRequestErrors**

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

**Cpa32U \_CpaCyKeyGenStats::numMgfKeyGenCompleted**

Total number of MGF key generation operations that completed successfully.

**Cpa32U \_CpaCyKeyGenStats::numMgfKeyGenCompletedErrors**

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

## 9.8 Define Documentation

**#define CPA\_CY\_KEY\_GEN\_SSL\_TLS\_RANDOM\_LEN\_IN\_BYTES**

SSL or TLS key generation random number length.

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

**See also:**

**`cpaCyKeyGenSsl()` `cpaCyKeyGenTls()`**

**#define CPA\_CY\_KEY\_GEN\_SSL\_TLS\_SEED\_LEN\_IN\_BYTES**

TLS seed length.

Defines the permitted SSL or TLS seed length in bytes that may be used with the `cpaCyKeyGenSsl` and `cpaCyKeyGenTls` functions. This is the length of the seed value.

**See also:**

**`cpaCyKeyGenSsl()` `cpaCyKeyGenTls()`**

## 9.9 Typedef Documentation

```
typedef enum _CpaCyKeySslOp CpaCyKeySslOp
```

SSL Operation Types.

Enumeration of the different SSL operations that can be specified in the CpaCyKeyGenSslOpData.

```
typedef struct _CpaCyKeyGenSslOpData CpaCyKeyGenSslOpData
```

SSL data for key generation functions.

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

SSL Master-Secret Derivation: sslOp = CPA\_CY\_KEY\_SSL\_OP\_MASTER\_SECRET\_DERIVE secret = pre-master secret key seed = client\_random + server\_random userLabel = NULL

SSL Key-Material Derivation: sslOp = CPA\_CY\_KEY\_SSL\_OP\_KEY\_MATERIAL\_DERIVE secret = master secret key seed = server\_random + client\_random userLabel = NULL

(note that the client/server random order is reversed from that used for Master-Secret Derivation)

Notes: 1. Each of the client and server random numbers need to be of length CPA\_CY\_KEY\_GEN\_SSL\_TLS\_RANDOM\_LEN\_IN\_BYTES. 2. In each of the above descriptions, + indicates concatenation. 3. The label used is predetermined by the SSL operation in line with the SSL 3.0 specification, and can be overridden by using a user defined operation CPA\_CY\_KEY\_SSL\_OP\_USER\_DEFINED and associated userLabel.

```
typedef enum _CpaCyKeyTlsOp CpaCyKeyTlsOp
```

TLS Operation Types.

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

```
typedef struct _CpaCyKeyGenTlsOpData CpaCyKeyGenTlsOpData
```

TLS data for key generation functions.

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

TLS Master-Secret Derivation: tlsOp = CPA\_CY\_KEY\_TLS\_OP\_MASTER\_SECRET\_DERIVE secret = pre-master secret key seed = client\_random + server\_random userLabel = NULL

TLS Key-Material Derivation: tlsOp = CPA\_CY\_KEY\_TLS\_OP\_KEY\_MATERIAL\_DERIVE secret = master secret key seed = server\_random + client\_random userLabel = NULL

(note that the client/server random order is reversed from that used for Master-Secret Derivation)

TLS Client finished/Server finished tag Derivation: tlsOp = CPA\_CY\_KEY\_TLS\_OP\_CLIENT\_FINISHED\_DERIVE (Client) or CPA\_CY\_KEY\_TLS\_OP\_SERVER\_FINISHED\_DERIVE (server) secret = master secret key seed = MD5(handshake\_messages) + SHA-1(handshake\_messages) userLabel = NULL

Notes: 1. Each of the client and server random seeds need to be of length CPA\_CY\_KEY\_GEN\_SSL\_TLS\_RANDOM\_LEN\_IN\_BYTES. 2. In each of the above descriptions, +

## 9.10 Enumeration Type Documentation

indicates concatenation. 3. The label used is predetermined by the TLS operation in line with the TLS 1.0 specification, and can be overridden by using a user defined operation CPA\_CY\_KEY\_TLS\_OP\_USER\_DEFINED and associated userLabel.

```
typedef struct _CpaCyKeyGenMgfOpData CpaCyKeyGenMgfOpData
```

Key Generation Mask Generation Function (MGF) Data.

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

```
typedef struct _CpaCyKeyGenStats CpaCyKeyGenStats
```

Key Generation Statistics.

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.

---

## 9.10 Enumeration Type Documentation

```
enum _CpaCyKeySslOp
```

SSL Operation Types.

Enumeration of the different SSL operations that can be specified in the CpaCyKeyGenSslOpData.

### Enumerator:

<i>CPA_CY_KEY_SSL_OP_MASTER_SECRET_DERIVE</i>	Derive the master secret.
<i>CPA_CY_KEY_SSL_OP_KEY_MATERIAL_DERIVE</i>	Derive the key material.
<i>CPA_CY_KEY_SSL_OP_USER_DEFINED</i>	User Defined Operation for custom labels.

```
enum _CpaCyKeyTlsOp
```

TLS Operation Types.

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

### Enumerator:

<i>CPA_CY_KEY_TLS_OP_MASTER_SECRET_DERIVE</i>	Derive the master secret using the TLS PRF.
<i>CPA_CY_KEY_TLS_OP_KEY_MATERIAL_DERIVE</i>	Derive the key material using the TLS PRF.
<i>CPA_CY_KEY_TLS_OP_CLIENT_FINISHED_DERIVE</i>	Derive the client finished tag using the TLS PRF.
<i>CPA_CY_KEY_TLS_OP_SERVER_FINISHED_DERIVE</i>	Derive the server finished tag using the TLS PRF.
<i>CPA_CY_KEY_TLS_OP_USER_DEFINED</i>	User Defined Operation for custom labels.

---

## 9.11 Function Documentation

```

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

```

SSL Key Generation Function.

This function is used for SSL key generation. The input seed is taken as a flat buffer and the generated key is returned to caller in a flat destination data buffer.

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

Yes when configured to operate in synchronous mode.

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

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

**Return values:**

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.

## 9.11 Function Documentation

*CPA\_STATUS\_INVALID\_PARAM* Invalid parameter passed in.  
*CPA\_STATUS\_RESOURCE* Error related to system resources.

### Precondition:

The component has been initialized via `cpaCyStartInstance` function.

### Postcondition:

None

### Note:

This function is only used to generate SSL keys from seed material.

### See also:

**CpaCyKeyGenSslOpData, CpaCyGenFlatBufCbFunc**

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

TLS Key Generation Function.

This function is used for TLS key generation. The input seed is taken as a flat buffer and the generated key is returned to caller in a flat destination data buffer.

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

Yes when configured to operate in synchronous mode.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pKeyGenCb</i>	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in]	<i>pKeyGenTlsOpData</i>	Structure containing all the data needed to perform the TLS key generation operation. The client code allocates the memory for this

structure. This component takes ownership of the memory until it is returned in the callback.

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

**Return values:**

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

**Precondition:**

The component has been initialized via *cpaCyStartInstance* function.

**Postcondition:**

None

**Note:**

This function is only used to generate TLS keys from seed material.

**See also:**

**CpaCyKeyGenTlsOpData, CpaCyGenFlatBufCbFunc**

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

Mask Generation Function.

This function is used for mask generation. 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

**Parameters:**

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pKeyGenCb</i>	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in]	<i>pKeyGenMgfOpData</i>	Structure containing all the data needed to perform the MGF key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pGeneratedMaskBuffer</i>	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 it's pOut parameter.

**Return values:**

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

**Precondition:**The component has been initialized via `cpaCyStartInstance` function.**Postcondition:**

None

**Note:**

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

**See also:****CpaCyKeyGenMgfOpData**, **CpaCyGenFlatBufCbFunc**

```
CpaStatus cpaCyKeyGenQueryStats ( const CpaInstanceHandle instanceHandle,
                                CpaCyKeyGenStats * pKeyGenStats
                                )
```

Key and Mask generation statistics specific to an instance.

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

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



## 9.11 Function Documentation

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

This function is synchronous and blocking.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

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

### Return values:

*CPA\_STATUS\_SUCCESS* Function executed successfully.  
*CPA\_STATUS\_FAIL* Function failed.  
*CPA\_STATUS\_INVALID\_PARAM* Invalid parameter passed in.  
*CPA\_STATUS\_RESOURCE* Error related to system resources.

### Precondition:

Component has been initialized.

### Postcondition:

None

### Note:

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

### See also:

**CpaCyKeyGenStats**

# 10 Crypto API Large Number.

## [Cryptographic API.]

Collaboration diagram for Crypto API Large Number.:



## 10.1 Detailed Description

These functions specify the Cryptographic API for Large Number Operations.

## 10.2 Data Structures

- struct **\_CpaCyLnModExpOpData**  
Modular Exponentiation Function Operation Data.
- struct **\_CpaCyLnModInvOpData**  
Modular Inversion Function Operation Data.
- struct **\_CpaCyLnStats**  
Look Aside Cryptographic large number Statistics.

## 10.3 Typedefs

- typedef **\_CpaCyLnModExpOpData CpaCyLnModExpOpData**  
Modular Exponentiation Function Operation Data.
- typedef **\_CpaCyLnModInvOpData CpaCyLnModInvOpData**  
Modular Inversion Function Operation Data.
- typedef **\_CpaCyLnStats CpaCyLnStats**  
Look Aside Cryptographic large number Statistics.

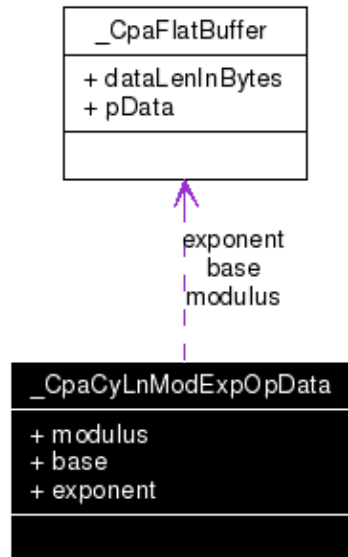
## 10.4 Functions

- **CpaStatus cpaCyLnModExp** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pLnModExpCb, void \*pCallbackTag, const **CpaCyLnModExpOpData** \*pLnModExpOpData, **CpaFlatBuffer** \*pResult)  
Function to for Modular Exponentiation operations.
- **CpaStatus cpaCyLnModInv** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pLnModInvCb, void \*pCallbackTag, const **CpaCyLnModInvOpData** \*pLnModInvOpData, **CpaFlatBuffer** \*pResult)  
Function for Modular Inversion operations.
- **CpaStatus cpaCyLnStatsQuery** (const **CpaInstanceHandle** instanceHandle, **CpaCyLnStats** \*pLnStats)  
Query statistics for large number operations.

## 10.5 Data Structure Documentation

### 10.5.1 \_CpaCyLnModExpOpData Struct Reference

Collaboration diagram for \_CpaCyLnModExpOpData:



#### 10.5.1.1 Detailed Description

Modular Exponentiation Function Operation Data.

This structure lists the different items that are required in the `cpaCyLnModExp` function. The client **MUST** allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback. The operation size in bits is equal to the size of whichever of the following is largest: the modulus, the base or the exponent.

**Note:**

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyLnModExp` function, and before it has been returned in the callback, undefined behavior will result. The values of the base, the exponent and the modulus **MUST** all be less than  $2^{4096}$ , and the modulus must not be equal to 0. All values in this structure are required to be in Most Significant Byte first order, e.g. `modulus.pData[0]` = MSB.

#### 10.5.1.2 Data Fields

- **CpaFlatBuffer modulus**  
Flat buffer containing a pointer to the modulus.
- **CpaFlatBuffer base**  
Flat buffer containing a pointer to the base.
- **CpaFlatBuffer exponent**  
Flat buffer containing a pointer to the exponent.

#### 10.5.1.3 Field Documentation

##### **CpaFlatBuffer \_CpaCyLnModExpOpData::modulus**

Flat buffer containing a pointer to the modulus.

### 10.5.1 \_CpaCyLnModExpOpData Struct Reference

This number may be up to 4096 bits in length. This number MUST be greater than zero.

#### CpaFlatBuffer \_CpaCyLnModExpOpData::base

Flat buffer containing a pointer to the base.

The maximum size of the number may be up to 4096 bits in length. The number MUST be from 0 to  $2^{sz}-1$  (where  $sz$  = the operation size).

#### CpaFlatBuffer \_CpaCyLnModExpOpData::exponent

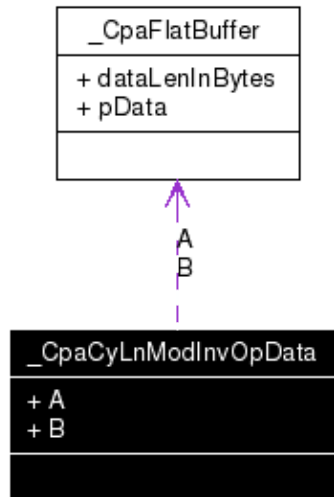
Flat buffer containing a pointer to the exponent.

The maximum size of the number may be up to 4096 bits in length. The number MUST be from 0 to  $2^{sz}-1$  (where  $sz$  = the operation size).

---

### 10.5.2 \_CpaCyLnModInvOpData Struct Reference

Collaboration diagram for \_CpaCyLnModInvOpData:



#### 10.5.2.1 Detailed Description

Modular Inversion Function Operation Data.

This structure lists the different items that are required in the `cpaCyLnModInv` 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 `CpaCyLnModExpCbFunc` structure. This structure is used to calculate:  $(1/pA) \bmod pB$ .

##### Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyLnModInv` function, and before it has been returned in the callback, undefined behavior will result. Note the value `pA` and the value `pB` MUST NOT both be even numbers and MUST be less than  $2^{4096}$ . All values in this structure are required to be in Most Significant Byte first order, e.g. `A.pData[0] = MSB`.

## 10.5.2 \_CpaCyLnModInvOpData Struct Reference

### 10.5.2.2 Data Fields

- **CpaFlatBuffer A**  
Flat buffer containing a pointer to the value that will be inverted.
- **CpaFlatBuffer B**  
Flat buffer containing a pointer to the value that will be used as the modulus.

### 10.5.2.3 Field Documentation

#### **CpaFlatBuffer \_CpaCyLnModInvOpData::A**

Flat buffer containing a pointer to the value that will be inverted.

This number may be up to 4096 bits in length. This number MUST NOT be zero and it MUST be co-prime with B.

#### **CpaFlatBuffer \_CpaCyLnModInvOpData::B**

Flat buffer containing a pointer to the value that will be used as the modulus.

This number MUST NOT be zero and it MUST be co-prime with A.

---

## 10.5.3 \_CpaCyLnStats Struct Reference

### 10.5.3.1 Detailed Description

Look Aside Cryptographic large number Statistics.

This structure contains statistics on the Look Aside Cryptographic large number operations. Statistics are set to zero when the component is initialized, and are collected per instance.

### 10.5.3.2 Data Fields

- **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.

### 10.5.3.3 Field Documentation

#### **Cpa32U \_CpaCyLnStats::numLnModExpRequests**

Total number of successful large number modular exponentiation requests.

#### **Cpa32U \_CpaCyLnStats::numLnModExpRequestErrors**

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

#### **Cpa32U \_CpaCyLnStats::numLnModExpCompleted**

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

#### **Cpa32U \_CpaCyLnStats::numLnModExpCompletedErrors**

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

#### **Cpa32U \_CpaCyLnStats::numLnModInvRequests**

Total number of successful large number modular inversion requests.

#### **Cpa32U \_CpaCyLnStats::numLnModInvRequestErrors**

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

#### **Cpa32U \_CpaCyLnStats::numLnModInvCompleted**

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

#### **Cpa32U \_CpaCyLnStats::numLnModInvCompletedErrors**

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

---

## 10.6 Typedef Documentation

#### **typedef struct \_CpaCyLnModExpOpData CpaCyLnModExpOpData**

Modular Exponentiation Function Operation Data.

This structure lists the different items that are required in the cpaCyLnModExp function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback. The operation size in bits is equal to the size of whichever of the following is largest: the modulus, the base or the exponent.

**Note:**

If the client modifies or frees the memory referenced in this structure after it has been submitted to the cpaCyLnModExp function, and before it has been returned in the callback, undefined behavior will result. The values of the base, the exponent and the modulus MUST all be less than  $2^{4096}$ , and the modulus must not be equal to 0. All values in this structure are required to be in Most Significant Byte first order, e.g. modulus.pData[0] = MSB.

#### **typedef struct \_CpaCyLnModInvOpData CpaCyLnModInvOpData**

Modular Inversion Function Operation Data.

## 10.6 Typedef Documentation

This structure lists the different items that are required in the `cpaCyLnModInv` 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 `CpaCyLnModExpCbFunc` structure. This structure is used to calculate:  $(1/pA) \bmod pB$ .

### Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyLnModInv` function, and before it has been returned in the callback, undefined behavior will result. Note the value `pA` and the value `pB` **MUST NOT** both be even numbers and **MUST** be less than  $2^{4096}$ . All values in this structure are required to be in Most Significant Byte first order, e.g. `A.pData[0] = MSB`.

```
typedef struct _CpaCyLnStats CpaCyLnStats
```

Look Aside Cryptographic large number Statistics.

This structure contains statistics on the Look Aside Cryptographic large number operations. Statistics are set to zero when the component is initialized, and are collected per instance.

## 10.7 Function Documentation

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

Function to for Modular Exponentiation operations.

This function may be used for modular exponentiation. It calculates:  $\text{result} = (\text{base} \wedge \text{exponent}) \bmod \text{modulus}$ .

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pLnModExpCb</i>	Pointer to callback function to be invoked when the operation is complete.
[in]	<i>pCallbackTag</i>	Opaque User Data for this specific call. Will be returned unchanged in the callback.

## 10.7 Function Documentation

[in] <i>pLnModExpOpData</i>	Structure containing all the data needed to perform the LN modular exponentiation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out] <i>pResult</i>	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 it's pOut parameter.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized.

### Postcondition:

None

### Note:

When pLnModExpCb is non null, an asynchronous callback of type CpaCyLnModExpCbFunc is generated in response to this function call. Any errors generated during processing are reported in the structure returned in the callback.

### See also:

**CpaCyLnModExpOpData, CpaCyGenFlatBufCbFunc**

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

Function for Modular Inversion operations.

This function may be used for modular Inversion. It calculates:  $\text{result} = (1/A) \bmod B$ .

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Reentrant:



## 10.7 Function Documentation

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pLnModInvCb</i>	Pointer to callback function to be invoked when the operation is complete.
[in]	<i>pCallbackTag</i>	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in]	<i>pLnModInvOpData</i>	Structure containing all the data needed to perform the LN modular inversion operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pResult</i>	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 it's pOut parameter.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized.

### Postcondition:

None

### Note:

When pLnModInvCb is non null, an asynchronous callback of type CpaCyLnModInvCbFunc is generated in response to this function call. Any errors generated during processing are reported in the structure returned in the callback.

### See also:

**CpaCyLnModInvOpData, CpaCyGenFlatBufCbFunc**

```
CpaStatus cpaCyLnStatsQuery ( const CpaInstanceHandle instanceHandle,  
                             CpaCyLnStats * pLnStats  
                             )
```

Query statistics for large number operations.

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

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

### Context:

## 10.7 Function Documentation

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

### Assumptions:

None

### Side-Effects:

None

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

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

### Return values:

*CPA\_STATUS\_SUCCESS* Function executed successfully.  
*CPA\_STATUS\_FAIL* Function failed.  
*CPA\_STATUS\_INVALID\_PARAM* Invalid parameter passed in.  
*CPA\_STATUS\_RESOURCE* Error related to system resources.

### Precondition:

Acceleration Services unit has been initialized.

### Postcondition:

None

### Note:

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

### See also:

**CpaCyLnStats**

# 11 Prime Number Test API.

## [Cryptographic API.]

Collaboration diagram for Prime Number Test API.:



## 11.1 Detailed Description

These functions specify the API for the prime number test operations.

For prime number generation, this API SHOULD be used in conjunction with the Random Number Generation API.

## 11.2 Data Structures

- struct **\_CpaCyPrimeTestOpData**  
Prime Test Operation Data.
- struct **\_CpaCyPrimeStats**  
Prime Number Test Statistics.

## 11.3 Typedefs

- typedef **\_CpaCyPrimeTestOpData CpaCyPrimeTestOpData**  
Prime Test Operation Data.
- typedef **\_CpaCyPrimeStats CpaCyPrimeStats**  
Prime Number Test Statistics.
- typedef void(\* **CpaCyPrimeTestCbFunc** )(void \*pCallbackTag, **CpaStatus** status, void \*pOpData, **CpaBoolean** testPassed)  
Definition of callback function invoked for cpaCyPrimeTest requests.

## 11.4 Functions

- **CpaStatus cpaCyPrimeTest** (const **CpaInstanceHandle** instanceHandle, const **CpaCyPrimeTestCbFunc** pCb, void \*pCallbackTag, const **CpaCyPrimeTestOpData** \*pOpData, **CpaBoolean** \*pTestPassed)  
Prime Number Test Function.

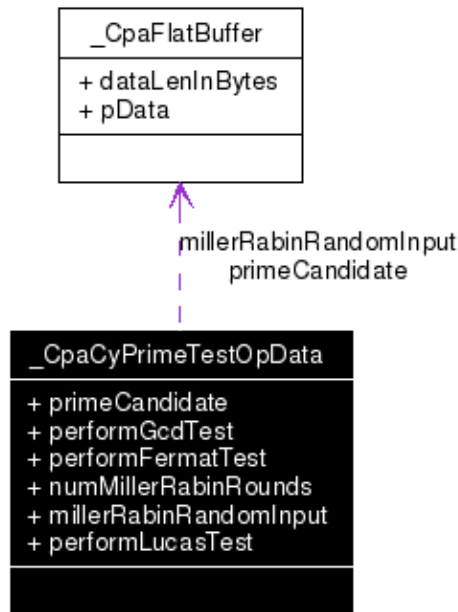
---

## 11.5 Data Structure Documentation

### 11.5.1 \_CpaCyPrimeTestOpData Struct Reference

Collaboration diagram for \_CpaCyPrimeTestOpData:

### 11.5.1 \_CpaCyPrimeTestOpData Struct Reference



#### 11.5.1.1 Detailed Description

Prime Test Operation Data.

This structure contains the operation data for the `cpaCyPrimeTest` function. The client **MUST** allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

All numbers **MUST** be stored in big-endian order.

**Note:**

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

**See also:**

**`cpaCyPrimeTest()`**

#### 11.5.1.2 Data Fields

- **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.
- **CpaFlatBuffer millerRabinRandomInput**  
Flat buffer containing a pointer to an array of random numbers for Miller Rabin Primality

### 11.5.1 \_CpaCyPrimeTestOpData Struct Reference

Tests.

- **CpaBoolean performLucasTest**

An CPA\_TRUE value means perform a Lucas Primality Test.

#### 11.5.1.3 Field Documentation

**CpaFlatBuffer \_CpaCyPrimeTestOpData::primeCandidate**

The prime number candidate to test.

**CpaBoolean \_CpaCyPrimeTestOpData::performGcdTest**

A value of CPA\_TRUE means perform a GCD Primality Test.

**CpaBoolean \_CpaCyPrimeTestOpData::performFermatTest**

A value of CPA\_TRUE means perform a Fermat Primality Test.

**Cpa32U \_CpaCyPrimeTestOpData::numMillerRabinRounds**

Number of Miller Rabin Primality Test rounds.

Set to 0 to perform zero Miller Rabin tests. The maximum number of rounds supported is 50.

**CpaFlatBuffer \_CpaCyPrimeTestOpData::millerRabinRandomInput**

Flat buffer containing a pointer to an array of random numbers for Miller Rabin Primality Tests.

The size of the array MUST match the requested number of rounds. Each random number MUST be greater than 1 and less than the prime candidate - 1. Each random number MUST match the prime candidate in size, with leading zeroes as necessary.

**CpaBoolean \_CpaCyPrimeTestOpData::performLucasTest**

An CPA\_TRUE value means perform a Lucas Primality Test.

---

### 11.5.2 \_CpaCyPrimeStats Struct Reference

#### 11.5.2.1 Detailed Description

Prime Number Test Statistics.

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.

#### 11.5.2.2 Data Fields

- **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**

### 11.5.2 \_CpaCyPrimeStats Struct Reference

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

#### 11.5.2.3 Field Documentation

##### **Cpa32U \_CpaCyPrimeStats::numPrimeTestRequests**

Total number of successful prime number test requests.

##### **Cpa32U \_CpaCyPrimeStats::numPrimeTestRequestErrors**

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

##### **Cpa32U \_CpaCyPrimeStats::numPrimeTestCompleted**

Total number of prime number test operations that completed successfully.

##### **Cpa32U \_CpaCyPrimeStats::numPrimeTestCompletedErrors**

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

##### **Cpa32U \_CpaCyPrimeStats::numPrimeTestFailures**

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

---

## 11.6 Typedef Documentation

##### **typedef struct \_CpaCyPrimeTestOpData CpaCyPrimeTestOpData**

Prime Test Operation Data.

This structure contains the operation data for the cpaCyPrimeTest function. The client MUST allocate the memory for this structure and the items pointed to by this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the callback function.

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

All numbers MUST be stored in big-endian order.

##### **Note:**

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

##### **See also:**

**cpaCyPrimeTest()**

##### **typedef struct \_CpaCyPrimeStats CpaCyPrimeStats**

Prime Number Test Statistics.

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.

## 11.6 Typedef Documentation

```
typedef void(* CpaCyPrimeTestCbFunc)(void *pCallbackTag, CpaStatus status, void *pOpData,  
CpaBoolean testPassed)
```

Definition of callback function invoked for cpaCyPrimeTest requests.

This is the prototype for the cpaCyPrimeTest callback function.

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in] <i>pCallbackTag</i>	User-supplied value to help identify request.
[in] <i>status</i>	Status of the operation. Valid values are CPA_STATUS_SUCCESS and CPA_STATUS_FAIL.
[in] <i>pOpData</i>	Opaque pointer to the Operation data pointer supplied in request.
[in] <i>testPassed</i>	A value of CPA_TRUE means the prime candidate is probably prime.

### Return values:

None

### Precondition:

Component has been initialized.

### Postcondition:

None

### Note:

None

### See also:

**cpaCyPrimeTest()**

---

## 11.7 Function Documentation

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

## 11.7 Function Documentation

### Prime Number Test Function.

This function will test probabilistically if a number is prime. Refer to ANSI X9.80 2005 for details. The primality result will be returned in the asynchronous callback.

The following combination of GCD, Fermat, Miller-Rabin, and Lucas testing is supported: (up to 1x GCD) + (up to 1x Fermat) + (up to 50x Miller-Rabin rounds) + (up to 1x Lucas) For example: (1x GCD) + (25x Miller-Rabin) + (1x Lucas); (1x GCD) + (1x Fermat); (50x Miller-rabin);

Tests are always performed in order of increasing complexity, for example GCD first, then Fermat, then Miller-Rabin, and finally Lucas.

For all of the primality tests, the following prime number sizes (in bits) are supported: 160, 512, 768, 1024, 1536, 2048, 3072, 4096.

Candidate prime numbers MUST match these sizes accordingly, with leading zeroes present where necessary.

When this prime number test is used in conjunction with combined Miller-Rabin and Lucas tests, it may be used as a means of performing a self test operation on the random data generator.

A response status of ok (pass == CPA\_TRUE) means all requested primality tests passed, and the prime candidate is probably prime (the exact probability depends on the primality tests requested). A response status of not ok (pass == CPA\_FALSE) means one of the requested primality tests failed (the prime candidate has been found to be composite).

#### Context:

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

#### Assumptions:

None

#### Side-Effects:

None

#### Blocking:

Yes when configured to operate in synchronous mode.

#### Reentrant:

No

#### Thread-safe:

Yes

#### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pCb</i>	Callback function pointer. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	User-supplied value to help identify request.
[in]	<i>pOpData</i>	Structure containing all the data needed to perform the operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pTestPassed</i>	A value of CPA_TRUE means the prime candidate is probably prime.



## 11.7 Function Documentation

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized via `cpaCyStartInstance` function.

### Postcondition:

None

### Note:

When `pCb` is non-NULL an asynchronous callback of type `CpaCyPrimeTestCbFunc` is generated in response to this function call. \* For optimal performance, data pointers SHOULD be 8-byte aligned.

### See also:

**`CpaCyPrimeTestOpData`, `CpaCyPrimeTestCbFunc`**

# 12 Random Bit/Number Generation API.

## [Cryptographic API.]

Collaboration diagram for Random Bit/Number Generation API.:



## 12.1 Detailed Description

These functions specify the API for the Cryptographic Random Bit and Random number generation.

## 12.2 Data Structures

- struct **\_CpaCyRandStats**  
Random Data Generator Statistics.
- struct **\_CpaCyRandGenOpData**  
Random Bit/Number Generation Data.
- struct **\_CpaCyRandSeedOpData**  
Random Generator Seed Data.

## 12.3 Defines

- #define **CPA\_CY\_RAND\_SEED\_LEN\_IN\_BYTES**  
Random Bit/Number Generator Seed Length.

## 12.4 Typedefs

- typedef **\_CpaCyRandStats CpaCyRandStats**  
Random Data Generator Statistics.
- typedef **\_CpaCyRandGenOpData CpaCyRandGenOpData**  
Random Bit/Number Generation Data.
- typedef **\_CpaCyRandSeedOpData CpaCyRandSeedOpData**  
Random Generator Seed Data.

## 12.5 Functions

- **CpaStatus cpaCyRandGen** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenFlatBufCbFunc** pRandGenCb, void \*pCallbackTag, const **CpaCyRandGenOpData** \*pRandGenOpData, **CpaFlatBuffer** \*pRandData)  
Random Bits or Number Generation Function.
  - **CpaStatus cpaCyRandSeed** (const **CpaInstanceHandle** instanceHandle, const **CpaCyGenericCbFunc** pRandSeedCb, void \*pCallbackTag, const **CpaCyRandSeedOpData** \*pSeedOpData)  
Random Data Generator Seed Function.
  - **CpaStatus cpaCyRandQueryStats** (const **CpaInstanceHandle** instanceHandle, **CpaCyRandStats** \*pRandStats)  
Query random number statistics specific to an instance.
-

## 12.6 Data Structure Documentation

### 12.6.1 \_CpaCyRandStats Struct Reference

#### 12.6.1.1 Detailed Description

Random Data Generator Statistics.

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

#### 12.6.1.2 Data Fields

- **Cpa32U numRandNumRequests**  
Total number of successful random number generation requests.
- **Cpa32U numRandNumRequestErrors**  
Total number of random number generation requests that had an error and could not be processed.
- **Cpa32U numRandNumCompleted**  
Total number of random number operations that completed successfully.
- **Cpa32U numRandNumCompletedErrors**  
Total number of random number operations that could not be completed successfully due to errors.
- **Cpa32U numRandBitRequests**  
Total number of successful random bit generation requests.
- **Cpa32U numRandBitRequestErrors**  
Total number of random bit generation requests that had an error and could not be processed.
- **Cpa32U numRandBitCompleted**  
Total number of random bit operations that completed successfully.
- **Cpa32U numRandBitCompletedErrors**  
Total number of random bit operations that could not be completed successfully due to errors.
- **Cpa32U numNumSeedRequests**  
Total number of seed operations requests.
- **Cpa32U numRandSeedCompleted**  
Total number of seed operations completed.
- **Cpa32U numNumSeedErrors**  
Total number of seed operation errors.

#### 12.6.1.3 Field Documentation

##### **Cpa32U \_CpaCyRandStats::numRandNumRequests**

Total number of successful random number generation requests.

##### **Cpa32U \_CpaCyRandStats::numRandNumRequestErrors**

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

##### **Cpa32U \_CpaCyRandStats::numRandNumCompleted**

Total number of random number operations that completed successfully.

##### **Cpa32U \_CpaCyRandStats::numRandNumCompletedErrors**

## 12.6.1 \_CpaCyRandStats Struct Reference

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

### **Cpa32U \_CpaCyRandStats::numRandBitRequests**

Total number of successful random bit generation requests.

### **Cpa32U \_CpaCyRandStats::numRandBitRequestErrors**

Total number of random bit generation requests that had an error and could not be processed.

### **Cpa32U \_CpaCyRandStats::numRandBitCompleted**

Total number of random bit operations that completed successfully.

### **Cpa32U \_CpaCyRandStats::numRandBitCompletedErrors**

Total number of random bit operations that could not be completed successfully due to errors.

### **Cpa32U \_CpaCyRandStats::numNumSeedRequests**

Total number of seed operations requests.

### **Cpa32U \_CpaCyRandStats::numRandSeedCompleted**

Total number of seed operations completed.

### **Cpa32U \_CpaCyRandStats::numNumSeedErrors**

Total number of seed operation errors.

---

## 12.6.2 \_CpaCyRandGenOpData Struct Reference

### 12.6.2.1 Detailed Description

Random Bit/Number Generation Data.

This structure lists the different items that are required in the cpaCyRandGen function. The client **MUST** allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

#### **Note:**

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

### 12.6.2.2 Data Fields

- **CpaBoolean generateBits**  
When set to CPA\_TRUE then the cpaCyRandGen function will generate random bits which will comply with the ANSI X9.82 Part 1 specification.
- **Cpa32U lenInBytes**  
Specifies the length in bytes of the data returned.

### 12.6.2.3 Field Documentation

#### **CpaBoolean \_CpaCyRandGenOpData::generateBits**

## 12.6.2 \_CpaCyRandGenOpData Struct Reference

When set to CPA\_TRUE then the cpaCyRandGen function will generate random bits which will comply with the ANSI X9.82 Part 1 specification.

When set to CPA\_FALSE random numbers will be produced from the random bits generated by the hardware. This will be spec compliant in terms of the probability of the random nature of the number returned.

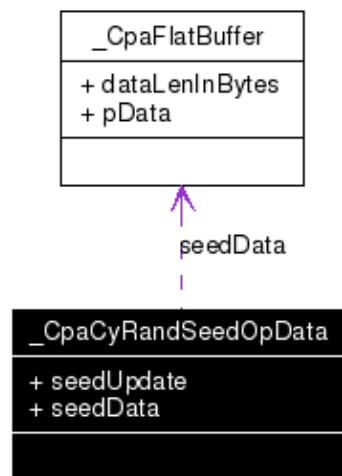
### Cpa32U \_CpaCyRandGenOpData::lenInBytes

Specifies the length in bytes of the data returned.

If the data returned is a random number, then it is implicit that the random number will fall into the following range: Expressed mathematically, the range is  $[2^{(\text{lenInBytes} * 8 - 1)} \text{ to } 2^{(\text{lenInBytes} * 8)} - 1]$ . This is equivalent to "1000...0000" to "1111...1111" which requires  $(\text{lenInBytes} * 8)$  bits to represent. The maximum number of random bytes that can be requested is 65535 bytes.

## 12.6.3 \_CpaCyRandSeedOpData Struct Reference

Collaboration diagram for \_CpaCyRandSeedOpData:



### 12.6.3.1 Detailed Description

Random Generator Seed Data.

This structure lists the different items that required in the cpaCyRandSeed function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

#### Note:

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

### 12.6.3.2 Data Fields

- **CpaBoolean seedUpdate**

When set to CPA\_TRUE then the cpaCyRandSeed function will update (combine) the specified seed with the stored seed.

### 12.6.3 \_CpaCyRandSeedOpData Struct Reference

- **CpaFlatBuffer seedData**

Data for use in either seeding or performing a seed update.

#### 12.6.3.3 Field Documentation

##### **CpaBoolean \_CpaCyRandSeedOpData::seedUpdate**

When set to CPA\_TRUE then the cpaCyRandSeed function will update (combine) the specified seed with the stored seed.

When set to CPA\_FALSE, the cpaCyRandSeed function will completely discard all existing entropy in the hardware and replace with the specified seed.

##### **CpaFlatBuffer \_CpaCyRandSeedOpData::seedData**

Data for use in either seeding or performing a seed update.

The data that is pointed to are random bits and as such do not have an endian order. For optimal performance the data SHOULD be 8-byte aligned. The length of the seed data is in bytes. This MUST currently be equal to CPA\_CY\_RAND\_SEED\_LEN\_IN\_BYTES.

---

## 12.7 Define Documentation

##### **#define CPA\_CY\_RAND\_SEED\_LEN\_IN\_BYTES**

Random Bit/Number Generator Seed Length.

Defines the permitted seed length in bytes that may be used with the cpaCyRandSeed function.

**See also:**

**cpaCyRandSeed**

---

## 12.8 Typedef Documentation

##### **typedef struct \_CpaCyRandStats CpaCyRandStats**

Random Data Generator Statistics.

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

##### **typedef struct \_CpaCyRandGenOpData CpaCyRandGenOpData**

Random Bit/Number Generation Data.

This structure lists the different items that are required in the cpaCyRandGen function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

**Note:**

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

```
typedef struct _CpaCyRandSeedOpData CpaCyRandSeedOpData
```

Random Generator Seed Data.

This structure lists the different items that required in the `cpaCyRandSeed` function. The client **MUST** allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned with the callback.

**Note:**

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

## 12.9 Function Documentation

```
CpaStatus cpaCyRandGen ( const CpaInstanceHandle      instanceHandle,
                        const CpaCyGenFlatBufCbFunc pRandGenCb,
                        void *                       pCallbackTag,
                        const CpaCyRandGenOpData *  pRandGenOpData,
                        CpaFlatBuffer *            pRandData
                        )
```

Random Bits or Number Generation Function.

This function is used to request the generation of random bits or a random number. The generated data and the length of the data will be returned to the caller in an asynchronous callback function. If random number generation is selected, the random bits generated by the hardware will be converted to a random number that is compliant to the ANSI X9.82 Part 1 specification.

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

Yes when configured to operate in synchronous mode.

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pRandGenCb</i>	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	

	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in] <i>pRandGenOpData</i>	Structure containing all the data needed to perform the random bit/number operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out] <i>pRandData</i>	Pointer to the memory allocated by the client where the random data will be written to. For optimal performance, the data pointed to SHOULD be 8-byte aligned. There is no endianness associated with the random data. On invocation the callback function will contain this parameter in it's pOut parameter.

**Return values:**

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources. One reason may be for an entropy test failing.

**Precondition:**

The component has been initialized via `cpaCyStartInstance` function.

**Postcondition:**

None

**Note:**

When `pRandGenCb` is non-NULL an asynchronous callback of type `CpaCyRandGenCbFunc` is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. Entropy testing and reseeding are performed automatically by this function.

**See also:**

**`CpaCyGenFlatBufCbFunc`, `CpaCyRandGenOpData`, `cpaCyRandSeed()`.**

```
CpaStatus cpaCyRandSeed (  const CpaInstanceHandle instanceHandle,
                           const
                           CpaCyGenericCbFunc      pRandSeedCb,
                           void *                    pCallbackTag,
                           const
                           CpaCyRandSeedOpData * pSeedOpData
                           )
```

Random Data Generator Seed Function.

This function is used to either seed or perform a seed update on the random data generator. Replacing the seed with a user supplied seed value, or performing a seed update are completely optional operations. If seeding is specified, it has the effect of disregarding all existing entropy within the random data generator and replacing with the specified seed. If performing a seed update, then the specified seed is mixed into the stored seed. The seed length MUST be equal to `CPA_CY_RANDOM_SEED_LEN_IN_BYTES`.

**Context:**

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



## 12.9 Function Documentation

### Assumptions:

None

### Side-Effects:

None

### Blocking:

Yes when configured to operate in synchronous mode.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

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

### Return values:

- CPA\_STATUS\_SUCCESS* Function executed successfully.
- CPA\_STATUS\_FAIL* Function failed.
- CPA\_STATUS\_RETRY* Resubmit the request.
- CPA\_STATUS\_INVALID\_PARAM* Invalid parameter passed in.
- CPA\_STATUS\_RESOURCE* Error related to system resources.

### Precondition:

The component has been initialized via *cpaCyStartInstance* function.

### Postcondition:

None

### Note:

When *pRandSeedCb* is non-NULL an asynchronous callback of type *CpaCyRandSeedCbFunc* is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. Entropy testing and reseeding are performed automatically by the *cpaCyRandGen* function.

### See also:

***CpaCyGenericCbFunc***, ***CpaCyRandSeedOpData***, ***cpaCyRandGen()***

```
CpaStatus cpaCyRandQueryStats ( const CpaInstanceHandle instanceHandle,  
                                CpaCyRandStats * pRandStats  
                                )
```

Query random number statistics specific to an instance.

This function will query a specific instance for random number statistics. The user MUST allocate the *CpaCyRandStats* structure and pass the reference to that into this function call. This function will write the

## 12.9 Function Documentation

statistic results into the passed in CpaCyRandStats structure.

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

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

This function is synchronous and blocking.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

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

### Return values:

*CPA\_STATUS\_SUCCESS* Function executed successfully.  
*CPA\_STATUS\_FAIL* Function failed.  
*CPA\_STATUS\_INVALID\_PARAM* Invalid parameter passed in.

### Precondition:

Component has been initialized.

### Postcondition:

None

### Note:

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

### See also:

**CpaCyRandStats**

# 13 Public Key Encryption RSA API.

## [Cryptographic API.]

Collaboration diagram for Public Key Encryption RSA API.:



## 13.1 Detailed Description

These functions specify the API for Public Key Encryption (cryptography) RSA operations.

The PKCS #1 V2.1 specification is supported, however the support is limited to "two-prime" mode. RSA multi-prime is not supported.

## 13.2 Data Structures

- struct **\_CpaCyRsaPublicKey**  
RSA Public Key Structure.
- struct **\_CpaCyRsaPrivateKeyRep1**  
RSA Private Key Structure For Representation 1.
- struct **\_CpaCyRsaPrivateKeyRep2**  
RSA Private Key Structure For Representation 2.
- struct **\_CpaCyRsaPrivateKey**  
RSA Private Key Structure.
- struct **\_CpaCyRsaKeyGenOpData**  
RSA Key Generation Data.
- struct **\_CpaCyRsaEncryptOpData**  
RSA Encryption Primitive Operation Data.
- struct **\_CpaCyRsaDecryptOpData**  
RSA Decryption Primitive Operation Data.
- struct **\_CpaCyRsaStats**  
RSA Statistics.

## 13.3 Typedefs

- typedef enum **\_CpaCyRsaVersion CpaCyRsaVersion**  
RSA Version.
- typedef **\_CpaCyRsaPublicKey CpaCyRsaPublicKey**  
RSA Public Key Structure.
- typedef **\_CpaCyRsaPrivateKeyRep1 CpaCyRsaPrivateKeyRep1**  
RSA Private Key Structure For Representation 1.
- typedef **\_CpaCyRsaPrivateKeyRep2 CpaCyRsaPrivateKeyRep2**  
RSA Private Key Structure For Representation 2.
- typedef enum **\_CpaCyRsaPrivateKeyRepType CpaCyRsaPrivateKeyRepType**  
RSA private key representation type.
- typedef **\_CpaCyRsaPrivateKey CpaCyRsaPrivateKey**  
RSA Private Key Structure.
- typedef **\_CpaCyRsaKeyGenOpData CpaCyRsaKeyGenOpData**  
RSA Key Generation Data.
- typedef **\_CpaCyRsaEncryptOpData CpaCyRsaEncryptOpData**

### 13.3 Typedefs

- RSA Encryption Primitive Operation Data.  
typedef **\_CpaCyRsaDecryptOpData CpaCyRsaDecryptOpData**  
RSA Decryption Primitive Operation Data.
- typedef **\_CpaCyRsaStats CpaCyRsaStats**  
RSA Statistics.
- typedef void(\* **CpaCyRsaKeyGenCbFunc** )(void \*pCallbackTag, **CpaStatus** status, void \*pKeyGenOpData, **CpaCyRsaPrivateKey** \*pPrivateKey, **CpaCyRsaPublicKey** \*pPublicKey)  
Definition of the RSA key generation callback function.

### 13.4 Enumerations

- enum **\_CpaCyRsaVersion** { **CPA\_CY\_RSA\_VERSION\_TWO\_PRIME** }  
RSA Version.
- enum **\_CpaCyRsaPrivateKeyRepType** {  
    **CPA\_CY\_RSA\_PRIVATE\_KEY\_REP\_TYPE\_1**,  
    **CPA\_CY\_RSA\_PRIVATE\_KEY\_REP\_TYPE\_2**  
}  
RSA private key representation type.

### 13.5 Functions

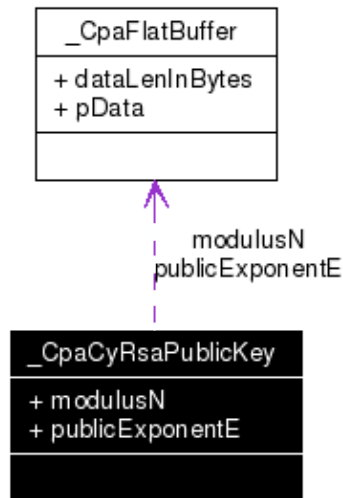
- **CpaStatus cpaCyRsaGenKey** (const **CpaInstanceHandle** instanceHandle, const **CpaCyRsaKeyGenCbFunc** pRsaKeyGenCb, void \*pCallbackTag, const **CpaCyRsaKeyGenOpData** \*pKeyGenOpData, **CpaCyRsaPrivateKey** \*pPrivateKey, **CpaCyRsaPublicKey** \*pPublicKey)  
Generate RSA keys.
  - **CpaStatus 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.
  - **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.
  - **CpaStatus cpaCyRsaQueryStats** (const **CpaInstanceHandle** instanceHandle, **CpaCyRsaStats** \*pRsaStats)  
Query statistics for a specific RSA instance.
- 

## 13.6 Data Structure Documentation

### 13.6.1 \_CpaCyRsaPublicKey Struct Reference

Collaboration diagram for **\_CpaCyRsaPublicKey**:

### 13.6.1 \_CpaCyRsaPublicKey Struct Reference



#### 13.6.1.1 Detailed Description

RSA Public Key Structure.

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

#### 13.6.1.2 Data Fields

- **CpaFlatBuffer modulusN**  
The modulus (n).
- **CpaFlatBuffer publicExponentE**  
The public exponent (e).

#### 13.6.1.3 Field Documentation

##### CpaFlatBuffer \_CpaCyRsaPublicKey::modulusN

The modulus (n).

For key generation operations, the client MUST allocate the memory for this parameter and it's value is generated. For encrypt operations this parameter is an input.

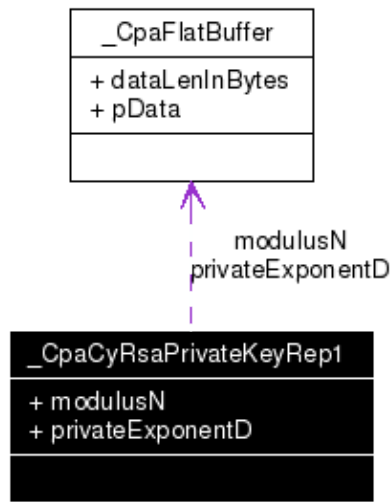
##### CpaFlatBuffer \_CpaCyRsaPublicKey::publicExponentE

The public exponent (e).

N.B. This value is not generated by this interface. It MUST be specified by the client. For key generation operations this pointer will be assigned to the public exponent parameter from the CpaCyRsaKeyGenOpData structure. For other operations this is an input.

## 13.6.2 \_CpaCyRsaPrivateKeyRep1 Struct Reference

Collaboration diagram for \_CpaCyRsaPrivateKeyRep1:



### 13.6.2.1 Detailed Description

RSA Private Key Structure For Representation 1.

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

### 13.6.2.2 Data Fields

- **CpaFlatBuffer modulusN**  
The modulus (n).
- **CpaFlatBuffer privateExponentD**  
The private exponent (d).

### 13.6.2.3 Field Documentation

#### CpaFlatBuffer \_CpaCyRsaPrivateKeyRep1::modulusN

The modulus (n).

For key generation operations the memory MUST be allocated by the client and the value is generated. For other operations this is an input. Permitted lengths are: 1024 bits (128 bytes) 1536 bits (192 bytes), 2048 bits (256 bytes), 3072 bits (384 bytes) or 4096 bits (512 bytes).

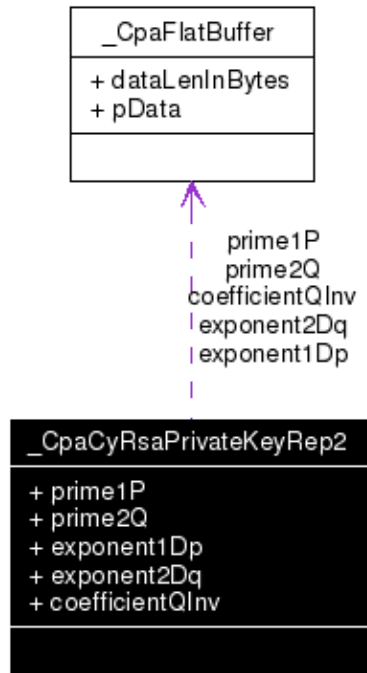
#### CpaFlatBuffer \_CpaCyRsaPrivateKeyRep1::privateExponentD

The private exponent (d).

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

### 13.6.3 \_CpaCyRsaPrivateKeyRep2 Struct Reference

Collaboration diagram for \_CpaCyRsaPrivateKeyRep2:



#### 13.6.3.1 Detailed Description

RSA Private Key Structure For Representation 2.

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

#### 13.6.3.2 Data Fields

- **CpaFlatBuffer prime1P**  
The first large prime ( $p$ ).
- **CpaFlatBuffer prime2Q**  
The second large prime ( $q$ ).
- **CpaFlatBuffer exponent1Dp**  
The first factor CRT exponent ( $dP$ ).
- **CpaFlatBuffer exponent2Dq**  
The second factor CRT exponent ( $dQ$ ).
- **CpaFlatBuffer coefficientQInv**  
The (first) Chinese Remainder Theorem (CRT) coefficient ( $qInv$ ).

#### 13.6.3.3 Field Documentation

##### CpaFlatBuffer \_CpaCyRsaPrivateKeyRep2::prime1P

The first large prime ( $p$ ).

#### 13.6.4 \_CpaCyRsaPrivateKey Struct Reference

##### **CpaFlatBuffer \_CpaCyRsaPrivateKeyRep2::prime2Q**

The second large prime (q).

##### **CpaFlatBuffer \_CpaCyRsaPrivateKeyRep2::exponent1Dp**

The first factor CRT exponent (dP).

$d \bmod (p-1)$ .

##### **CpaFlatBuffer \_CpaCyRsaPrivateKeyRep2::exponent2Dq**

The second factor CRT exponent (dQ).

$d \bmod (q-1)$ .

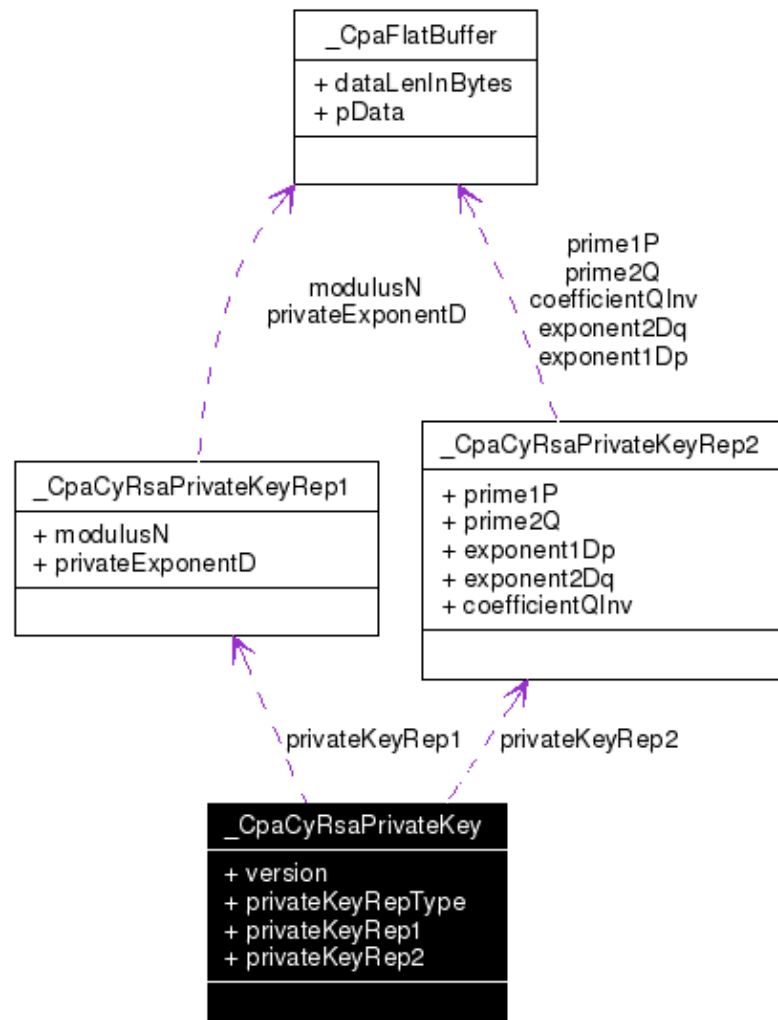
##### **CpaFlatBuffer \_CpaCyRsaPrivateKeyRep2::coefficientQInv**

The (first) Chinese Remainder Theorem (CRT) coefficient (qInv).

(inverse of q) mod p.

#### 13.6.4 \_CpaCyRsaPrivateKey Struct Reference

Collaboration diagram for \_CpaCyRsaPrivateKey:





## 13.6.4 \_CpaCyRsaPrivateKey Struct Reference

### 13.6.4.1 Detailed Description

RSA Private Key Structure.

This structure contains the two representations that can be used for describing the RSA private key. The `privateKeyRepType` will be used to identify which representation is to be used. Typically, using the second representation results in faster decryption operations.

### 13.6.4.2 Data Fields

- **CpaCyRsaVersion version**  
Indicates the version of the PKCS #1 specification that is supported.
- **CpaCyRsaPrivateKeyRepType privateKeyRepType**  
This value is used to identify which of the private key representation types in this structure is relevant.
- **CpaCyRsaPrivateKeyRep1 privateKeyRep1**  
This is the first representation of the RSA private key as defined in the PKCS #1 V2.1 specification.
- **CpaCyRsaPrivateKeyRep2 privateKeyRep2**  
This is the second representation of the RSA private key as defined in the PKCS #1 V2.1 specification.

### 13.6.4.3 Field Documentation

#### **CpaCyRsaVersion \_CpaCyRsaPrivateKey::version**

Indicates the version of the PKCS #1 specification that is supported.

N.B. This applies to both representations.

#### **CpaCyRsaPrivateKeyRepType \_CpaCyRsaPrivateKey::privateKeyRepType**

This value is used to identify which of the private key representation types in this structure is relevant.

When performing key generation operations for Type 2 representations, memory must also be allocated for the type 1 representations, and values for both will be returned.

#### **CpaCyRsaPrivateKeyRep1 \_CpaCyRsaPrivateKey::privateKeyRep1**

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

For key generation operations the memory for this structure is allocated by the client and the specific values are generated. For other operations this is an input parameter

#### **CpaCyRsaPrivateKeyRep2 \_CpaCyRsaPrivateKey::privateKeyRep2**

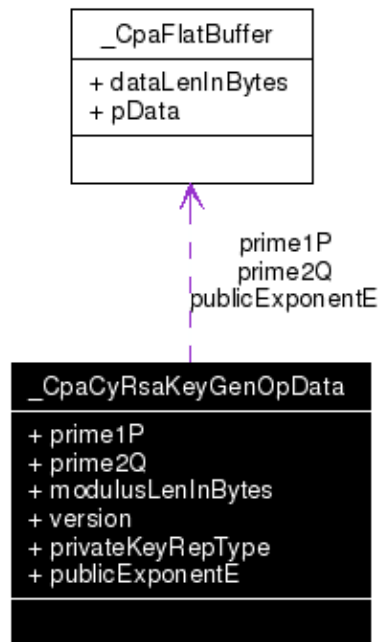
This is the second representation of the RSA private key as defined in the PKCS #1 V2.1 specification.

For key generation operations the memory for this structure is allocated by the client and the specific values are generated. For other operations this is an input parameter.

---

## 13.6.5 \_CpaCyRsaKeyGenOpData Struct Reference

Collaboration diagram for \_CpaCyRsaKeyGenOpData:



### 13.6.5.1 Detailed Description

RSA Key Generation Data.

This structure lists the different items that are required in the `cpaCyRsaGenKey` function. The client **MUST** allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the `CpaCyRsaKeyGenCbFunc` callback function.

**Note:**

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyRsaGenKey` function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. `prime1P.pData[0] = MSB`.

The following limitations on the permutations of the supported bit lengths of `p`, `q` and `n` apply:  $\{p, q, n\} = \{512, 512, 1024\}$  or  $\{768, 768, 1536\}$  or  $\{1024, 1024, 2048\}$  or  $\{1536, 1536, 3072\}$  or  $\{2048, 2048, 4096\}$ .

### 13.6.5.2 Data Fields

- **CpaFlatBuffer prime1P**  
A large random prime number (`p`).
- **CpaFlatBuffer prime2Q**  
A large random prime number (`q`).
- **Cpa32U modulusLenInBytes**  
The bit length of the modulus (`n`).
- **CpaCyRsaVersion version**  
Indicates the version of the PKCS #1 specification that is supported.
- **CpaCyRsaPrivateKeyRepType privateKeyRepType**

### 13.6.5 \_CpaCyRsaKeyGenOpData Struct Reference

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

- **CpaFlatBuffer publicExponentE**  
The public exponent (e).

#### 13.6.5.3 Field Documentation

##### **CpaFlatBuffer \_CpaCyRsaKeyGenOpData::prime1P**

A large random prime number (p).

This MUST be created by the client. Permitted bit lengths are: 512, 1024 or 2048 bits. Limitations apply - refer to the description above for details.

##### **CpaFlatBuffer \_CpaCyRsaKeyGenOpData::prime2Q**

A large random prime number (q).

This MUST be created by the client. Permitted bit lengths are: 512, 768, 1024, 1536 or 2048 bits. Limitations apply - refer to the description above for details. If the private key representation type is 2, then this pointer will be assigned to the relevant structure member of the representation 2 private key.

##### **Cpa32U \_CpaCyRsaKeyGenOpData::modulusLenInBytes**

The bit length of the modulus (n).

This is the modulus length for both the private and public keys. The length of the modulus N parameter for the private key representation 1 structure and the public key structures will be assigned to this value. References to the strength of RSA actually refer to this bit length. Recommended minimum is 1024 bits. Permitted lengths are: 1024 bits (128 bytes), 1536 bits (192 bytes), 2048 bits (256 bytes), 3072 bits (384 bytes) or 4096 bits (512 bytes). Limitations apply - refer to description above for details.

##### **CpaCyRsaVersion \_CpaCyRsaKeyGenOpData::version**

Indicates the version of the PKCS #1 specification that is supported.

N.B. This applies to both representations.

##### **CpaCyRsaPrivateKeyRepType \_CpaCyRsaKeyGenOpData::privateKeyRepType**

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

##### **CpaFlatBuffer \_CpaCyRsaKeyGenOpData::publicExponentE**

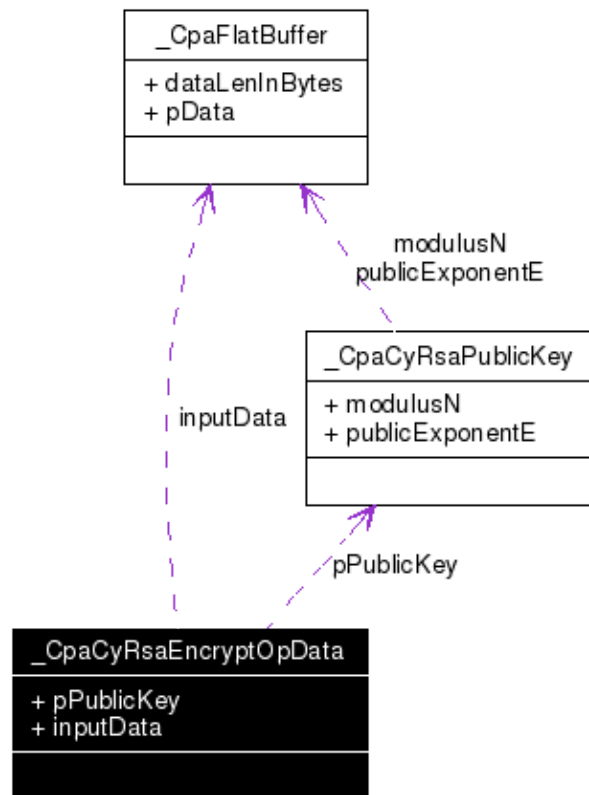
The public exponent (e).

---

### 13.6.6 \_CpaCyRsaEncryptOpData Struct Reference

Collaboration diagram for \_CpaCyRsaEncryptOpData:

### 13.6.6 \_CpaCyRsaEncryptOpData Struct Reference



#### 13.6.6.1 Detailed Description

RSA Encryption Primitive Operation Data.

This structure lists the different items that are required in the `cpaCyRsaEncrypt` function. As the RSA encryption primitive and verification primitive operations are mathematically identical this structure may also be used to perform an RSA verification primitive operation. When performing an RSA encryption primitive operation, the input data is the message and the output data is the cipher text. When performing an RSA verification primitive operation, the input data is the signature and the output data is the message. The client **MUST** allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the `CpaCyRsaEncryptCbFunc` callback function.

#### Note:

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyRsaEncrypt` function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. `inputData.pData[0] = MSB`.

#### 13.6.6.2 Data Fields

- **CpaCyRsaPublicKey \* pPublicKey**  
Pointer to the public key.
- **CpaFlatBuffer inputData**  
The input data that the RSA encryption primitive operation is performed on.

## 13.6.6 \_CpaCyRsaEncryptOpData Struct Reference

### 13.6.6.3 Field Documentation

#### **CpaCyRsaPublicKey\* \_CpaCyRsaEncryptOpData::pPublicKey**

Pointer to the public key.

#### **CpaFlatBuffer \_CpaCyRsaEncryptOpData::inputData**

The input data that the RSA encryption primitive operation is performed on.

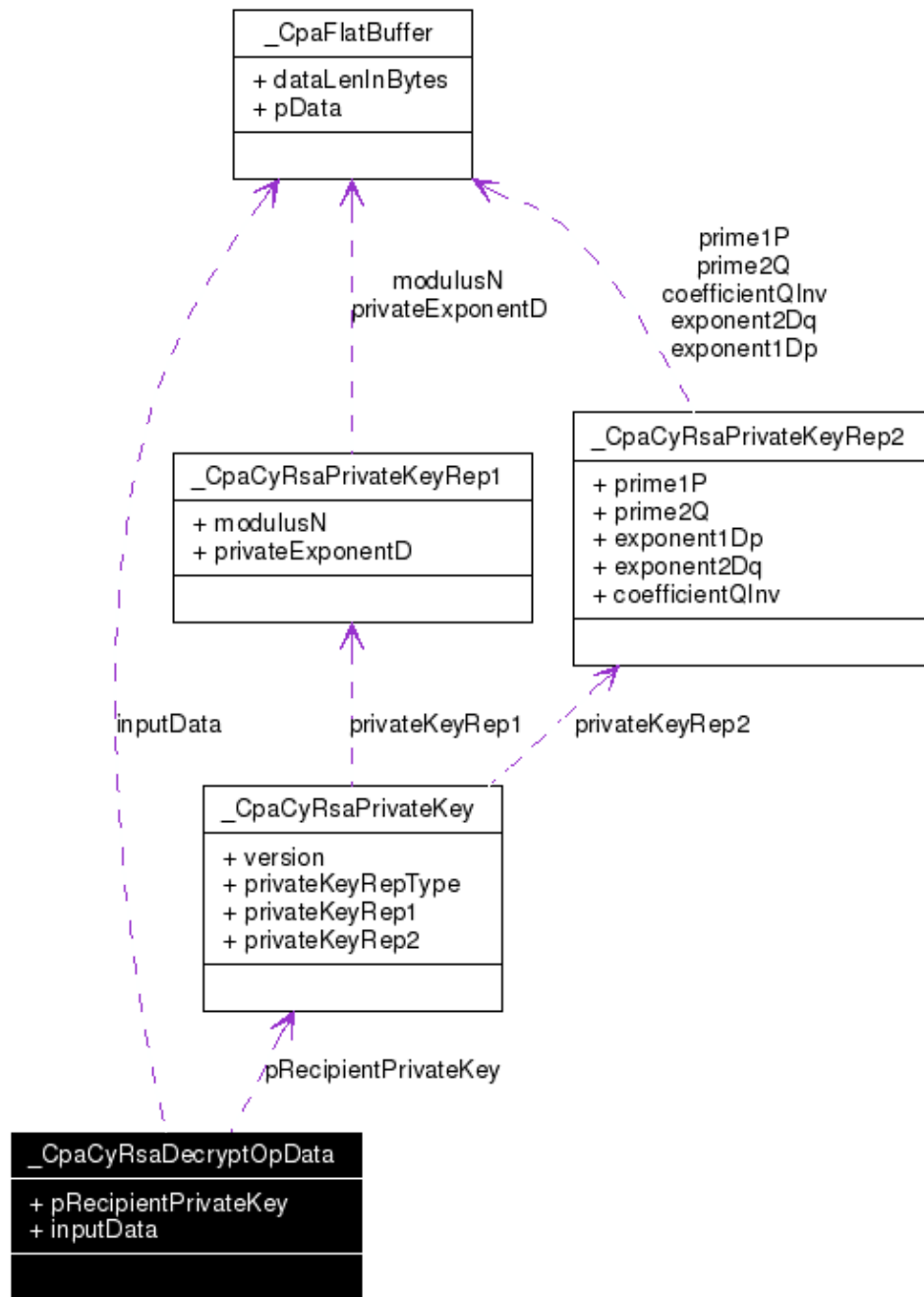
The data pointed to is an integer that MUST be in big- endian order. The value MUST be between 0 and the modulus  $n - 1$ .

---

## 13.6.7 \_CpaCyRsaDecryptOpData Struct Reference

Collaboration diagram for \_CpaCyRsaDecryptOpData:

### 13.6.7 \_CpaCyRsaDecryptOpData Struct Reference



#### 13.6.7.1 Detailed Description

RSA Decryption Primitive Operation Data.

This structure lists the different items that are required in the `cpaCyRsaDecrypt` function. As the RSA decryption primitive and signature primitive operations are mathematically identical this structure may also be used to perform an RSA signature primitive operation. When performing an RSA decryption primitive operation, the input data is the cipher text and the output data is the message text. When performing an RSA signature primitive operation, the input data is the message and the output data is the signature. The client **MUST** allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the `CpaCyRsaDecryptCbFunc` callback function.

### 13.6.7 \_CpaCyRsaDecryptOpData Struct Reference

#### 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.

#### 13.6.7.2 Data Fields

- **CpaCyRsaPrivateKey \* pRecipientPrivateKey**  
Pointer to the recipient's RSA private key.
- **CpaFlatBuffer inputData**  
The input data that the RSA decryption primitive operation is performed on.

#### 13.6.7.3 Field Documentation

##### **CpaCyRsaPrivateKey\* \_CpaCyRsaDecryptOpData::pRecipientPrivateKey**

Pointer to the recipient's RSA private key.

##### **CpaFlatBuffer \_CpaCyRsaDecryptOpData::inputData**

The input data that the RSA decryption primitive operation is performed on.

The data pointed to is an integer that MUST be in big- endian order. The value MUST be between 0 and the modulus  $n - 1$ .

---

### 13.6.8 \_CpaCyRsaStats Struct Reference

#### 13.6.8.1 Detailed Description

RSA Statistics.

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

#### 13.6.8.2 Data Fields

- **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.

### 13.6.8 \_CpaCyRsaStats Struct Reference

- **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.

#### 13.6.8.3 Field Documentation

##### **Cpa32U \_CpaCyRsaStats::numRsaKeyGenRequests**

Total number of successful RSA key generation requests.

##### **Cpa32U \_CpaCyRsaStats::numRsaKeyGenRequestErrors**

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

##### **Cpa32U \_CpaCyRsaStats::numRsaKeyGenCompleted**

Total number of RSA key generation operations that completed successfully.

##### **Cpa32U \_CpaCyRsaStats::numRsaKeyGenCompletedErrors**

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

##### **Cpa32U \_CpaCyRsaStats::numRsaEncryptRequests**

Total number of successful RSA encrypt operation requests.

##### **Cpa32U \_CpaCyRsaStats::numRsaEncryptRequestErrors**

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

##### **Cpa32U \_CpaCyRsaStats::numRsaEncryptCompleted**

Total number of RSA encrypt operations that completed successfully.

##### **Cpa32U \_CpaCyRsaStats::numRsaEncryptCompletedErrors**

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

##### **Cpa32U \_CpaCyRsaStats::numRsaDecryptRequests**

Total number of successful RSA decrypt operation requests.

##### **Cpa32U \_CpaCyRsaStats::numRsaDecryptRequestErrors**

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

##### **Cpa32U \_CpaCyRsaStats::numRsaDecryptCompleted**

Total number of RSA decrypt operations that completed successfully.

##### **Cpa32U \_CpaCyRsaStats::numRsaDecryptCompletedErrors**

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

---



## 13.7 Typedef Documentation

```
typedef enum _CpaCyRsaVersion CpaCyRsaVersion
```

RSA Version.

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

**Note:**

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

```
typedef struct _CpaCyRsaPublicKey CpaCyRsaPublicKey
```

RSA Public Key Structure.

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

```
typedef struct _CpaCyRsaPrivateKeyRep1 CpaCyRsaPrivateKeyRep1
```

RSA Private Key Structure For Representation 1.

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

```
typedef struct _CpaCyRsaPrivateKeyRep2 CpaCyRsaPrivateKeyRep2
```

RSA Private Key Structure For Representation 2.

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

```
typedef enum _CpaCyRsaPrivateKeyRepType CpaCyRsaPrivateKeyRepType
```

RSA private key representation type.

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

```
typedef struct _CpaCyRsaPrivateKey CpaCyRsaPrivateKey
```

RSA Private Key Structure.

This structure contains the two representations that can be used for describing the RSA private key. The privateKeyRepType will be used to identify which representation is to be used. Typically, using the second representation results in faster decryption operations.

```
typedef struct _CpaCyRsaKeyGenOpData CpaCyRsaKeyGenOpData
```

RSA Key Generation Data.

This structure lists the different items that are required in the cpaCyRsaGenKey function. The client MUST allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the CpaCyRsaKeyGenCbFunc callback function.

**Note:**

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyRsaGenKey` function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. `prime1P.pData[0] = MSB`.

The following limitations on the permutations of the supported bit lengths of `p`, `q` and `n` apply: `{p, q, n} = {512, 512, 1024}` or `{768, 768, 1536}` or `{1024, 1024, 2048}` or `{1536, 1536, 3072}` or `{2048, 2048, 4096}`.

```
typedef struct _CpaCyRsaEncryptOpData CpaCyRsaEncryptOpData
```

RSA Encryption Primitive Operation Data.

This structure lists the different items that are required in the `cpaCyRsaEncrypt` function. As the RSA encryption primitive and verification primitive operations are mathematically identical this structure may also be used to perform an RSA verification primitive operation. When performing an RSA encryption primitive operation, the input data is the message and the output data is the cipher text. When performing an RSA verification primitive operation, the input data is the signature and the output data is the message. The client **MUST** allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the `CpaCyRsaEncryptCbFunc` callback function.

**Note:**

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyRsaEncrypt` function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. `inputData.pData[0] = MSB`.

```
typedef struct _CpaCyRsaDecryptOpData CpaCyRsaDecryptOpData
```

RSA Decryption Primitive Operation Data.

This structure lists the different items that are required in the `cpaCyRsaDecrypt` function. As the RSA decryption primitive and signature primitive operations are mathematically identical this structure may also be used to perform an RSA signature primitive operation. When performing an RSA decryption primitive operation, the input data is the cipher text and the output data is the message text. When performing an RSA signature primitive operation, the input data is the message and the output data is the signature. The client **MUST** allocate the memory for this structure. When the structure is passed into the function, ownership of the memory passes to the function. Ownership of the memory returns to the client when this structure is returned in the `CpaCyRsaDecryptCbFunc` callback function.

**Note:**

If the client modifies or frees the memory referenced in this structure after it has been submitted to the `cpaCyRsaDecrypt` function, and before it has been returned in the callback, undefined behavior will result. All values in this structure are required to be in Most Significant Byte first order, e.g. `inputData.pData[0] = MSB`.

```
typedef struct _CpaCyRsaStats CpaCyRsaStats
```

RSA Statistics.

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

```
typedef void(* CpaCyRsaKeyGenCbFunc)(void *pCallbackTag, CpaStatus status, void *pKeyGenOpData, CpaCyRsaPrivateKey *pPrivateKey, CpaCyRsaPublicKey *pPublicKey)
```

## 13.8 Enumeration Type Documentation

Definition of the RSA key generation callback function.

This is the prototype for the RSA key generation callback function. The callback function pointer is passed in as a parameter to the `cpaCyRsaGenKey` function. It will be invoked once the request has completed.

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

[in] <i>pCallbackTag</i>	Opaque value provided by user while making individual function calls.
[in] <i>status</i>	Status of the operation. Valid values are CPA_STATUS_SUCCESS and CPA_STATUS_FAIL.
[in] <i>pKeyGenOpData</i>	Structure with output params for callback.
[in] <i>pPrivateKey</i>	Structure which contains pointers to the memory into which the generated private key will be written.
[in] <i>pPublicKey</i>	Structure which contains pointers to the memory into which the generated public key will be written. The pointer to the public exponent (e) that is returned in this structure is equal to the input public exponent.

**Return values:**

None

**Precondition:**

Component has been initialized.

**Postcondition:**

None

**Note:**

None

**See also:**

`CpaCyRsaPrivateKey`, `CpaCyRsaPublicKey`, `cpaCyRsaGenKey()`

---

## 13.8 Enumeration Type Documentation

### enum `_CpaCyRsaVersion`

RSA Version.

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

**Note:**

## 13.9 Function Documentation

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

### Enumerator:

*CPA\_CY\_RSA\_VERSION\_TWO\_PRIME* The version supported is "two-prime".

### enum **\_CpaCyRsaPrivateKeyRepType**

RSA private key representation type.

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

### Enumerator:

*CPA\_CY\_RSA\_PRIVATE\_KEY\_REP\_TYPE\_1* The first representation of the RSA private key.

*CPA\_CY\_RSA\_PRIVATE\_KEY\_REP\_TYPE\_2* The second representation of the RSA private key.

---

## 13.9 Function Documentation

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

Generate RSA keys.

This function will generate private and public keys for RSA as specified in the PKCS #1 V2.1 standard. Both representation types of the private key may be generated.

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

Yes when configured to operate in synchronous mode.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in] *instanceHandle* Instance handle.  
[in] *pRsaKeyGenCb*

## 13.9 Function Documentation

		Pointer to the callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in]	<i>pKeyGenOpData</i>	Structure containing all the data needed to perform the RSA key generation operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pPrivateKey</i>	Structure which contains pointers to the memory into which the generated private key will be written.
[out]	<i>pPublicKey</i>	Structure which contains pointers to the memory into which the generated public key will be written. The pointer to the public exponent (e) that is returned in this structure is equal to the input public exponent.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized via `cpaCyStartInstance` function.

### Postcondition:

None

### Note:

When `pRsaKeyGenCb` is non-NULL, an asynchronous callback of type is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. For optimal performance, data pointers SHOULD be 8-byte aligned.

### See also:

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

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

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

This function will perform an RSA encryption primitive operation on the input data using the specified RSA public key. As the RSA encryption primitive and verification primitive operations are mathematically identical this function may also be used to perform an RSA verification primitive operation.

### Context:

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

### Assumptions:

## 13.9 Function Documentation

None

### Side-Effects:

None

### Blocking:

Yes when configured to operate in synchronous mode.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pRsaEncryptCb</i>	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in]	<i>pEncryptOpData</i>	Structure containing all the data needed to perform the RSA encryption operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pOutputData</i>	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. It's value will be between 0 and the modulus $n - 1$ . On invocation the callback function will contain this parameter in it's pOut parameter.

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized via `cpaCyStartInstance` function.

### Postcondition:

None

### Note:

When `pRsaEncryptCb` is non-NULL an asynchronous callback of type is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. For optimal performance, data pointers SHOULD be 8-byte aligned.

### See also:

**CpaCyGenFlatBufCbFunc CpaCyRsaEncryptOpData cpaCyRsaGenKey() cpaCyRsaDecrypt()**

```

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

```

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

This function will perform an RSA decryption primitive operation on the input data using the specified RSA private key. As the RSA decryption primitive and signing primitive operations are mathematically identical this function may also be used to perform an RSA signing primitive operation.

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

Yes when configured to operate in synchronous mode.

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pRsaDecryptCb</i>	Pointer to callback function to be invoked when the operation is complete. If this is set to a NULL value the function will operate synchronously.
[in]	<i>pCallbackTag</i>	Opaque User Data for this specific call. Will be returned unchanged in the callback.
[in]	<i>pDecryptOpData</i>	Structure containing all the data needed to perform the RSA decrypt operation. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[out]	<i>pOutputData</i>	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. It's value will be between 0 and the modulus n - 1. On invocation the callback function will contain this parameter in it's pOut parameter.

**Return values:**

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.

## 13.9 Function Documentation

*CPA\_STATUS\_INVALID\_PARAM* Invalid parameter passed in.  
*CPA\_STATUS\_RESOURCE* Error related to system resources.

### Precondition:

The component has been initialized via `cpaCyStartInstance` function.

### Postcondition:

None

### Note:

When `pRsaDecryptCb` is non-NULL an asynchronous callback is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. For optimal performance, data pointers SHOULD be 8-byte aligned.

### See also:

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

```
CpaStatus cpaCyRsaQueryStats ( const CpaInstanceHandle instanceHandle,  
                               CpaCyRsaStats * pRsaStats  
                               )
```

Query statistics for a specific RSA instance.

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

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

### Context:

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

### Assumptions:

None

### Side-Effects:

None

### Blocking:

This function is synchronous and blocking.

### Reentrant:

No

### Thread-safe:

Yes

### Parameters:

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

### Return values:

*CPA\_STATUS\_SUCCESS* Function executed successfully.



### 13.9 Function Documentation

*CPA\_STATUS\_FAIL* Function failed.  
*CPA\_STATUS\_INVALID\_PARAM* Invalid parameter passed in.

**Precondition:**

Component has been initialized.

**Postcondition:**

None

**Note:**

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

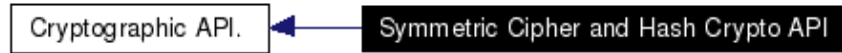
**See also:**

**CpaCyRsaStats**

# 14 Symmetric Cipher and Hash Crypto API

## [Cryptographic API.]

Collaboration diagram for Symmetric Cipher and Hash Crypto API:



## 14.1 Detailed Description

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

## 14.2 Data Structures

- struct **\_CpaCySymCipherSetupData**  
Symmetric Cipher Setup Data.
- struct **\_CpaCySymHashNestedModeSetupData**  
Hash Mode Nested Setup Data.
- struct **\_CpaCySymHashAuthModeSetupData**  
Hash Auth Mode Setup Data.
- struct **\_CpaCySymHashSetupData**  
Hash Setup Data.
- struct **\_CpaCySymSessionSetupData**  
Session Setup Data.
- struct **\_CpaCySymOpData**  
Cryptographic Component Operation Data.
- struct **\_CpaCySymStats**  
Cryptographic Component Statistics.

## 14.3 Typedefs

- typedef void \* **CpaCySymSessionCtx**  
Cryptographic component symmetric session context handle.
- typedef enum **\_CpaCySymPacketType CpaCySymPacketType**  
Packet type for the cpaCySymPerformOp function.
- typedef enum **\_CpaCySymOp CpaCySymOp**  
Types of operations supported by the cpaCySymPerformOp function.
- typedef enum **\_CpaCySymCipherAlgorithm CpaCySymCipherAlgorithm**  
Cipher algorithms.
- typedef enum **\_CpaCySymCipherDirection CpaCySymCipherDirection**  
Symmetric Cipher Direction.
- typedef **\_CpaCySymCipherSetupData CpaCySymCipherSetupData**  
Symmetric Cipher Setup Data.
- typedef enum **\_CpaCySymHashMode CpaCySymHashMode**  
Symmetric Hash mode.
- typedef enum **\_CpaCySymHashAlgorithm CpaCySymHashAlgorithm**  
Hash algorithms.
- typedef **\_CpaCySymHashNestedModeSetupData CpaCySymHashNestedModeSetupData**  
Hash Mode Nested Setup Data.
- typedef **\_CpaCySymHashAuthModeSetupData CpaCySymHashAuthModeSetupData**

### 14.3 Typedefs

- typedef **\_CpaCySymHashSetupData CpaCySymHashSetupData**  
Hash Auth Mode Setup Data.
- typedef enum **\_CpaCySymAlgChainOrder CpaCySymAlgChainOrder**  
Hash Setup Data.
- typedef enum **\_CpaCySymAlgChainOrder CpaCySymAlgChainOrder**  
Algorithm Chaining Operation Ordering.
- typedef **\_CpaCySymSessionSetupData CpaCySymSessionSetupData**  
Session Setup Data.
- typedef **\_CpaCySymOpData CpaCySymOpData**  
Cryptographic Component Operation Data.
- typedef **\_CpaCySymStats CpaCySymStats**  
Cryptographic Component Statistics.
- typedef void(\* **CpaCySymCbFunc** )(void \*pCallbackTag, **CpaStatus** status, const **CpaCySymOp** operationType, void \*pOpData, **CpaBufferList** \*pDstBuffer, **CpaBoolean** verifyResult)  
Definition of callback function.

### 14.4 Enumerations

- enum **\_CpaCySymPacketType** {  
    **CPA\_CY\_SYM\_PACKET\_TYPE\_FULL**,  
    **CPA\_CY\_SYM\_PACKET\_TYPE\_PARTIAL**,  
    **CPA\_CY\_SYM\_PACKET\_TYPE\_LAST\_PARTIAL**  
}  
Packet type for the cpaCySymPerformOp function.
- enum **\_CpaCySymOp** {  
    **CPA\_CY\_SYM\_OP\_CIPHER**,  
    **CPA\_CY\_SYM\_OP\_HASH**,  
    **CPA\_CY\_SYM\_OP\_ALGORITHM\_CHAINING**  
}  
Types of operations supported by the cpaCySymPerformOp function.
- enum **\_CpaCySymCipherAlgorithm** {  
    **CPA\_CY\_SYM\_CIPHER\_NULL**,  
    **CPA\_CY\_SYM\_CIPHER\_ARC4**,  
    **CPA\_CY\_SYM\_CIPHER\_AES\_ECB**,  
    **CPA\_CY\_SYM\_CIPHER\_AES\_CBC**,  
    **CPA\_CY\_SYM\_CIPHER\_AES\_CTR**,  
    **CPA\_CY\_SYM\_CIPHER\_AES\_CCM**,  
    **CPA\_CY\_SYM\_CIPHER\_AES\_GCM**,  
    **CPA\_CY\_SYM\_CIPHER\_DES\_ECB**,  
    **CPA\_CY\_SYM\_CIPHER\_DES\_CBC**,  
    **CPA\_CY\_SYM\_CIPHER\_3DES\_ECB**,  
    **CPA\_CY\_SYM\_CIPHER\_3DES\_CBC**,  
    **CPA\_CY\_SYM\_CIPHER\_3DES\_CTR**  
}  
Cipher algorithms.
- enum **\_CpaCySymCipherDirection** {  
    **CPA\_CY\_SYM\_CIPHER\_DIRECTION\_ENCRYPT**,  
    **CPA\_CY\_SYM\_CIPHER\_DIRECTION\_DECRYPT**  
}  
Symmetric Cipher Direction.
- enum **\_CpaCySymHashMode** {  
    **CPA\_CY\_SYM\_HASH\_MODE\_PLAIN**,  
    **CPA\_CY\_SYM\_HASH\_MODE\_AUTH**,  
    **CPA\_CY\_SYM\_HASH\_MODE\_NESTED**  
}  
Symmetric Hash mode.

## 14.4 Enumerations

- enum **\_CpaCySymHashAlgorithm** {  
    **CPA\_CY\_SYM\_HASH\_MD5**,  
    **CPA\_CY\_SYM\_HASH\_SHA1**,  
    **CPA\_CY\_SYM\_HASH\_SHA224**,  
    **CPA\_CY\_SYM\_HASH\_SHA256**,  
    **CPA\_CY\_SYM\_HASH\_SHA384**,  
    **CPA\_CY\_SYM\_HASH\_SHA512**,  
    **CPA\_CY\_SYM\_HASH\_AES\_XCBC**,  
    **CPA\_CY\_SYM\_HASH\_AES\_CCM**,  
    **CPA\_CY\_SYM\_HASH\_AES\_GCM**  
}  
    Hash algorithms.
- enum **\_CpaCySymAlgChainOrder** {  
    **CPA\_CY\_SYM\_ALG\_CHAIN\_ORDER\_HASH\_THEN\_CIPHER**,  
    **CPA\_CY\_SYM\_ALG\_CHAIN\_ORDER\_CIPHER\_THEN\_HASH**  
}  
    Algorithm Chaining Operation Ordering.

## 14.5 Functions

- **CpaStatus cpaCySymSessionCtxGetSize** (const **CpaInstanceHandle** instanceHandle, const **CpaCySymSessionSetupData** \*pSessionSetupData, **Cpa32U** \*pSessionCtxSizeInBytes)  
    Cryptographic Symmetric Session Context Size Get Function.
  - **CpaStatus cpaCySymInitSession** (const **CpaInstanceHandle** instanceHandle, const **CpaCySymCbFunc** pSymCb, const **CpaCySymSessionSetupData** \*pSessionSetupData, **CpaCySymSessionCtx** pSessionCtx)  
    Cryptographic Component Symmetric Session Initialization Function.
  - **CpaStatus cpaCySymRemoveSession** (const **CpaInstanceHandle** instanceHandle, **CpaCySymSessionCtx** pSessionCtx)  
    Cryptographic Component Symmetric Session Remove Function.
  - **CpaStatus cpaCySymPerformOp** (const **CpaInstanceHandle** instanceHandle, void \*pCallbackTag, const **CpaCySymOpData** \*pOpData, const **CpaBufferList** \*pSrcBuffer, **CpaBufferList** \*pDstBuffer, **CpaBoolean** \*pVerifyResult)  
    Cryptographic Component Symmetric Operation Perform Function.
  - **CpaStatus cpaCySymQueryStats** (const **CpaInstanceHandle** instanceHandle, **CpaCySymStats** \*pSymStats)  
    Query symmetric cryptographic statistics for a specific instance.
- 

## 14.6 Data Structure Documentation

### 14.6.1 \_CpaCySymCipherSetupData Struct Reference

#### 14.6.1.1 Detailed Description

Symmetric Cipher Setup Data.

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

#### 14.6.1.2 Data Fields

- **CpaCySymCipherAlgorithm cipherAlgorithm**  
    Cipher algorithm and mode.
- **Cpa32U cipherKeyLenInBytes**  
    Cipher key length in bytes.

## 14.6.1 \_CpaCySymCipherSetupData Struct Reference

- **Cpa8U \* pCipherKey**  
Cipher key.
- **CpaCySymCipherDirection cipherDirection**  
This parameter determines if the cipher operation is an encrypt or a decrypt operation.

### 14.6.1.3 Field Documentation

#### **CpaCySymCipherAlgorithm \_CpaCySymCipherSetupData::cipherAlgorithm**

Cipher algorithm and mode.

#### **Cpa32U \_CpaCySymCipherSetupData::cipherKeyLenInBytes**

Cipher key length in bytes.

For AES it can be 128 bits (16 bytes), 192 bits (24 bytes) or 256 bits (32 bytes). For the CCM mode of operation, the only supported key length is 128 bits (16 bytes).

#### **Cpa8U\* \_CpaCySymCipherSetupData::pCipherKey**

Cipher key.

#### **CpaCySymCipherDirection \_CpaCySymCipherSetupData::cipherDirection**

This parameter determines if the cipher operation is an encrypt or a decrypt operation.

---

## 14.6.2 \_CpaCySymHashNestedModeSetupData Struct Reference

### 14.6.2.1 Detailed Description

Hash Mode Nested Setup Data.

This structure contains data relating to a hash session in CPA\_CY\_SYM\_HASH\_MODE\_NESTED mode

### 14.6.2.2 Data Fields

- **Cpa8U \* pInnerPrefixData**  
A pointer to a buffer holding the Inner Prefix data.
- **Cpa32U innerPrefixLenInBytes**  
The inner prefix length in bytes.
- **CpaCySymHashAlgorithm outerHashAlgorithm**  
The hash algorithm used for the outer hash.
- **Cpa8U \* pOuterPrefixData**  
A pointer to a buffer holding the Outer Prefix data.
- **Cpa32U outerPrefixLenInBytes**  
The outer prefix length in bytes.

### 14.6.2.3 Field Documentation

#### **Cpa8U\* \_CpaCySymHashNestedModeSetupData::pInnerPrefixData**

A pointer to a buffer holding the Inner Prefix data.

For optimal performance the prefix data SHOULD be 8-byte aligned. This data is prepended to the data being hashed before the inner hash operation is performed.

## 14.6.2 \_CpaCySymHashNestedModeSetupData Struct Reference

### **Cpa32U \_CpaCySymHashNestedModeSetupData::innerPrefixLenInBytes**

The inner prefix length in bytes.

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

### **CpaCySymHashAlgorithm \_CpaCySymHashNestedModeSetupData::outerHashAlgorithm**

The hash algorithm used for the outer hash.

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

### **Cpa8U\* \_CpaCySymHashNestedModeSetupData::pOuterPrefixData**

A pointer to a buffer holding the Outer Prefix data.

For optimal performance the prefix data SHOULD be 8-byte aligned. This data is prepended to the output from the inner hash operation before the outer hash operation is performed.

### **Cpa32U \_CpaCySymHashNestedModeSetupData::outerPrefixLenInBytes**

The outer prefix length in bytes.

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

---

## 14.6.3 \_CpaCySymHashAuthModeSetupData Struct Reference

### 14.6.3.1 Detailed Description

Hash Auth Mode Setup Data.

This structure contains data relating to a hash session in CPA\_CY\_SYM\_HASH\_MODE\_AUTH mode

### 14.6.3.2 Data Fields

- **Cpa8U \* authKey**  
Authentication key pointer.
- **Cpa32U authKeyLenInBytes**  
Length of the authentication key in bytes.
- **Cpa32U aadLenInBytes**  
The length of the additional authenticated data (AAD) in bytes.

### 14.6.3.3 Field Documentation

#### **Cpa8U\* \_CpaCySymHashAuthModeSetupData::authKey**

Authentication key pointer.

#### **Cpa32U \_CpaCySymHashAuthModeSetupData::authKeyLenInBytes**

Length of the authentication key in bytes.

The key length MUST be less than or equal to the block size of the algorithm. It is the clients responsibility to ensure that the key length is compliant with the standard being used. For example RFC 2104, FIPS 198a. For the CCM mode of operation, the only supported key length is 128 bits (16 bytes).

### 14.6.3 \_CpaCySymHashAuthModeSetupData Struct Reference

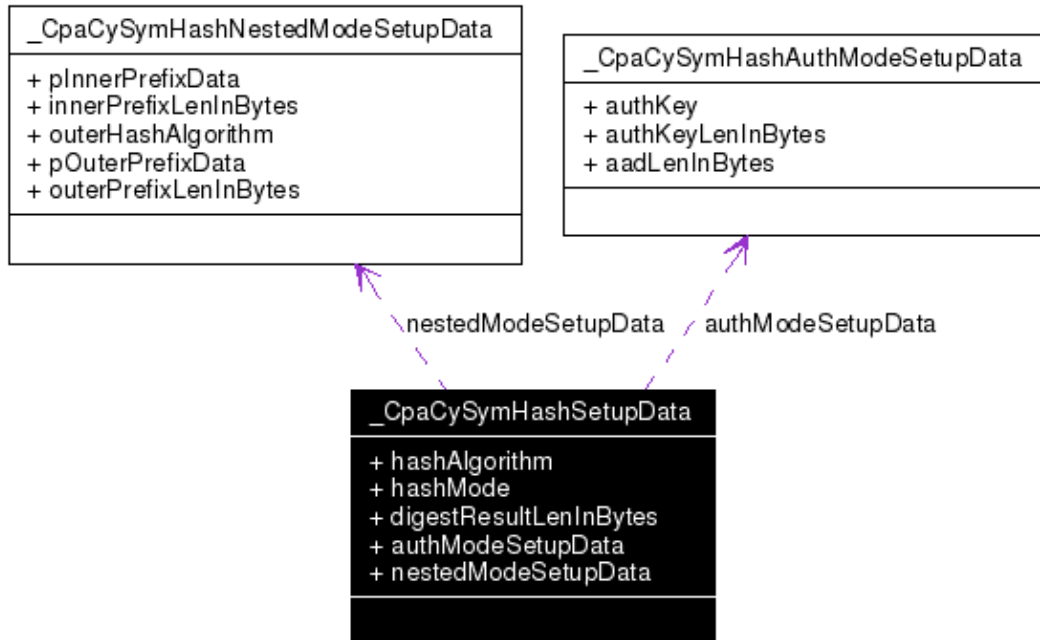
#### Cpa32U \_CpaCySymHashAuthModeSetupData::aadLenInBytes

The length of the additional authenticated data (AAD) in bytes.

This is only required for CCM and GCM modes of operation. For CCM, this is the length of the B blocks (including B0) that contain I(a) encoded, a itself, and any necessary padding. For GCM, this is the length of A. In all cases, the maximum permitted value is 240 bytes.

### 14.6.4 \_CpaCySymHashSetupData Struct Reference

Collaboration diagram for \_CpaCySymHashSetupData:



#### 14.6.4.1 Detailed Description

Hash Setup Data.

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

#### 14.6.4.2 Data Fields

- **CpaCySymHashAlgorithm hashAlgorithm**  
Hash algorithm.
- **CpaCySymHashMode hashMode**  
Mode of the hash operation.
- **Cpa32U digestResultLenInBytes**  
Length of the digest to be returned.
- **CpaCySymHashAuthModeSetupData authModeSetupData**  
Authentication Mode Setup Data.
- **CpaCySymHashNestedModeSetupData nestedModeSetupData**  
Nested Hash Mode Setup Data Only valid for mode `CPA_CY_SYM_MODE_HASH_NESTED`.

#### 14.6.4 \_CpaCySymHashSetupData Struct Reference

##### 14.6.4.3 Field Documentation

###### **CpaCySymHashAlgorithm \_CpaCySymHashSetupData::hashAlgorithm**

Hash algorithm.

For mode CPA\_CY\_SYM\_MODE\_HASH\_NESTED, this is the inner hash algorithm.

###### **CpaCySymHashMode \_CpaCySymHashSetupData::hashMode**

Mode of the hash operation.

Valid options include plain, auth or nested hash mode.

###### **Cpa32U \_CpaCySymHashSetupData::digestResultLenInBytes**

Length of the digest to be returned.

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

###### **CpaCySymHashAuthModeSetupData \_CpaCySymHashSetupData::authModeSetupData**

Authentication Mode Setup Data.

Only valid for mode CPA\_CY\_SYM\_MODE\_HASH\_AUTH

###### **CpaCySymHashNestedModeSetupData \_CpaCySymHashSetupData::nestedModeSetupData**

Nested Hash Mode Setup Data Only valid for mode CPA\_CY\_SYM\_MODE\_HASH\_NESTED.

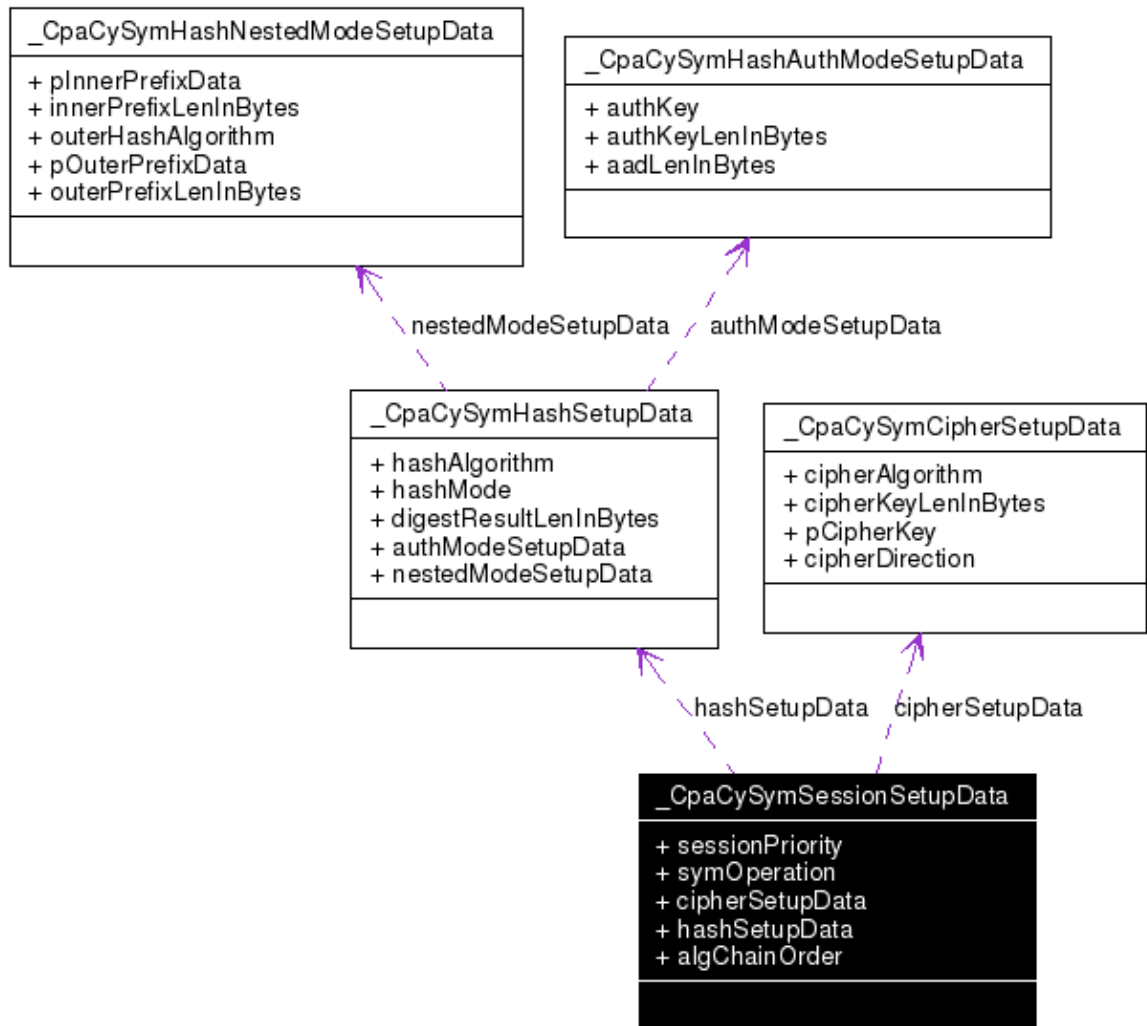
---

#### 14.6.5 \_CpaCySymSessionSetupData Struct Reference

Collaboration diagram for \_CpaCySymSessionSetupData:



#### 14.6.5 \_CpaCySymSessionSetupData Struct Reference



##### 14.6.5.1 Detailed Description

Session Setup Data.

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

##### 14.6.5.2 Data Fields

- **CpaCyPriority sessionPriority**  
Priority of this session.
- **CpaCySymOp symOperation**  
Operation: cipher, hash, auth cipher or chained.
- **CpaCySymCipherSetupData cipherSetupData**  
Cipher Setup Data for the session.
- **CpaCySymHashSetupData hashSetupData**  
Hash Setup Data for a session.
- **CpaCySymAlgChainOrder algChainOrder**  
If this operation data structure relates to an algorithm chaining session then this parameter determines the order the chained operations are performed in.

### 14.6.5.3 Field Documentation

#### **CpaCyPriority \_CpaCySymSessionSetupData::sessionPriority**

Priority of this session.

#### **CpaCySymOp \_CpaCySymSessionSetupData::symOperation**

Operation: cipher, hash, auth cipher or chained.

#### **CpaCySymCipherSetupData \_CpaCySymSessionSetupData::cipherSetupData**

Cipher Setup Data for the session.

This member is ignored for the CPA\_CY\_SYM\_OP\_HASH operation.

#### **CpaCySymHashSetupData \_CpaCySymSessionSetupData::hashSetupData**

Hash Setup Data for a session.

This member is ignored for the CPA\_CY\_SYM\_OP\_CIPHER operation

#### **CpaCySymAlgChainOrder \_CpaCySymSessionSetupData::algChainOrder**

If this operation data structure relates to an algorithm chaining session then this parameter determines the order the chained operations are performed in.

If this structure does not relate to an algorithm chaining session then this parameter will be ignored.

---

## 14.6.6 \_CpaCySymOpData Struct Reference

### 14.6.6.1 Detailed Description

Cryptographic Component Operation Data.

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

#### **See also:**

**CpaCySymPacketType**

#### **Note:**

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

### 14.6.6.2 Data Fields

- **CpaCySymSessionCtx pSessionCtx**  
Handle for the initialized session context.
- **CpaCySymPacketType packetType**  
Selects the perform operation packet type, i.e.
- **Cpa8U \* pIv**  
Initialization Vector or Counter.
- **Cpa32U ivLenInBytes**  
Cipher IV length in bytes.

#### 14.6.6 \_CpaCySymOpData Struct Reference

- **Cpa32U cryptoStartSrcOffsetInBytes**  
Starting point for cipher processing - given as number of bytes from start of data in the source buffer.
- **Cpa32U messageLenToCipherInBytes**  
The message length, in bytes, of the source buffer that the crypto operation will be computed on.
- **Cpa32U hashStartSrcOffsetInBytes**  
Starting point for hash processing - given as number of bytes from start of packet in source buffer.
- **Cpa32U messageLenToHashInBytes**  
The message length, in bytes, of the source buffer that the hash will be computed on.
- **Cpa8U \* pDigestResult**  
Pointer to the location where the digest result either exists or will be inserted.
- **Cpa8U \* pAdditionalAuthData**  
Pointer to Additional Authenticated Data (AAD) needed for authenticated cipher mechanisms - CCM and GCM.
- **CpaBoolean digestVerify**  
Compute the digest and compare it to the digest contained at the location pointed to by pDigestResult.

#### 14.6.6.3 Field Documentation

##### **CpaCySymSessionCtx \_CpaCySymOpData::pSessionCtx**

Handle for the initialized session context.

##### **CpaCySymPacketType \_CpaCySymOpData::packetType**

Selects the perform operation packet type, i.e.

Complete packet, a partial packet, or the final packet in a multi-part partial packet.

##### **Cpa8U\* \_CpaCySymOpData::plv**

Initialization Vector or Counter.

For block ciphers in CBC, CCM, and GCM mode this contains a pointer to the Initialization Vector (IV) value. For CCM this is the A0 block, while for GCM this is the Y0 block. For block ciphers in CTR mode this contains a pointer to the Counter. For optimum performance, the data pointed to SHOULD be 8-byte aligned. The IV/Counter will be updated after every partial crypto operation

##### **Cpa32U \_CpaCySymOpData::ivLenInBytes**

Cipher IV length in bytes.

Determines the amount of valid IV data pointed to by the plv parameter.

##### **Cpa32U \_CpaCySymOpData::cryptoStartSrcOffsetInBytes**

Starting point for cipher processing - given as number of bytes from start of data in the source buffer.

The result of the cipher operation will be written back into the output buffer starting at this location.

##### **Cpa32U \_CpaCySymOpData::messageLenToCipherInBytes**

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

### 14.6.7 \_CpaCySymStats Struct Reference

This must be a multiple to the block size if a block cipher is being used. This is also the same as the result length. 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).

#### **Cpa32U \_CpaCySymOpData::hashStartSrcOffsetInBytes**

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

Note: for CCM and GCM modes of operation set the pAdditionalAuthData field instead.

#### **Cpa32U \_CpaCySymOpData::messageLenToHashInBytes**

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

Note: There are limitations on this length for partial operations. Refer to the cpaCySymPerformOp function description for details. Note: for CCM and GCM modes of operation set the pAdditionalAuthData field instead. Note: On some implementations, this length may be limited to a 16-bit value (65535 bytes).

#### **Cpa8U\* \_CpaCySymOpData::pDigestResult**

Pointer to the location where the digest result either exists or will be inserted.

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 this pointer will be ignored for all but the final partial operation. NOTE: The digest result will overwrite any data at this location.

#### **Cpa8U\* \_CpaCySymOpData::pAdditionalAuthData**

Pointer to Additional Authenticated Data (AAD) needed for authenticated cipher mechanisms - CCM and GCM.

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.

#### **CpaBoolean \_CpaCySymOpData::digestVerify**

Compute the digest and compare it to the digest contained at the location pointed to by pDigestResult.

This option is only valid for full packets and for final partial packets. The number of bytes to be compared is indicated by the digest output length for the session. The digest computed will not be written back to the buffer. NOTE: This is not supported for hash mode CPA\_CY\_SYM\_HASH\_MODE\_NESTED

---

## 14.6.7 \_CpaCySymStats Struct Reference

### 14.6.7.1 Detailed Description

Cryptographic Component Statistics.

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

### 14.6.7.2 Data Fields

- **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 crypto operation requests.
- **Cpa32U numSymOpRequestErrors**  
Number of crypto operation requests that had an error and could not be processed.
- **Cpa32U numSymOpCompleted**  
Number of crypto operations that completed successfully.
- **Cpa32U numSymOpCompletedErrors**  
Number of crypto operations that could not be completed successfully due to errors.
- **Cpa32U numSymOpVerifyFailures**  
Number of crypto operations that completed successfully, but the result of the digest verification test was that it failed.

### 14.6.7.3 Field Documentation

#### **Cpa32U \_CpaCySymStats::numSessionsInitialized**

Number of session initialized.

#### **Cpa32U \_CpaCySymStats::numSessionsRemoved**

Number of sessions removed.

#### **Cpa32U \_CpaCySymStats::numSessionErrors**

Number of session initialized and removed errors.

#### **Cpa32U \_CpaCySymStats::numSymOpRequests**

Number of successful symmetric crypto operation requests.

#### **Cpa32U \_CpaCySymStats::numSymOpRequestErrors**

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

#### **Cpa32U \_CpaCySymStats::numSymOpCompleted**

Number of crypto operations that completed successfully.

#### **Cpa32U \_CpaCySymStats::numSymOpCompletedErrors**

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

#### **Cpa32U \_CpaCySymStats::numSymOpVerifyFailures**

Number of crypto operations that completed successfully, but the result of the digest verification test was that it failed.

N.B. This does not indicate an "error" condition.

---

## 14.7 Typedef Documentation

```
typedef void* CpaCySymSessionCtx
```

Cryptographic component symmetric session context handle.

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

```
typedef enum _CpaCySymPacketType CpaCySymPacketType
```

Packet type for the cpaCySymPerformOp function.

Enumeration which is used to indicate to the symmetric crypto perform function what type of packet the operation is required to be invoked on. The permitted types are a full packet, a partial packet, or the last part of a multi-part partial packet. 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:**

Partial packet processing is only supported for in-place cipher or in-place hash or in-place authentication operations. It does not apply to hash mode nested or algorithm chaining operations. The term "in-place operations" means that the result of the cipher or hash is written back into the source buffer.

**See also:**

**cpaCySymPerformOp()**

```
typedef enum _CpaCySymOp CpaCySymOp
```

Types of operations supported by the cpaCySymPerformOp function.

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

**See also:**

**cpaCySymPerformOp**

```
typedef enum _CpaCySymCipherAlgorithm CpaCySymCipherAlgorithm
```

Cipher algorithms.

This enumeration lists supported cipher algorithms and modes.

```
typedef enum _CpaCySymCipherDirection CpaCySymCipherDirection
```

Symmetric Cipher Direction.

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

```
typedef struct _CpaCySymCipherSetupData CpaCySymCipherSetupData
```

Symmetric Cipher Setup Data.

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

## 14.7 Typedef Documentation

typedef enum **\_CpaCySymHashMode CpaCySymHashMode**

Symmetric Hash mode.

This enum indicates the Hash Mode.

typedef enum **\_CpaCySymHashAlgorithm CpaCySymHashAlgorithm**

Hash algorithms.

This enumeration lists supported hash algorithms.

typedef struct **\_CpaCySymHashNestedModeSetupData CpaCySymHashNestedModeSetupData**

Hash Mode Nested Setup Data.

This structure contains data relating to a hash session in CPA\_CY\_SYM\_HASH\_MODE\_NESTED mode

typedef struct **\_CpaCySymHashAuthModeSetupData CpaCySymHashAuthModeSetupData**

Hash Auth Mode Setup Data.

This structure contains data relating to a hash session in CPA\_CY\_SYM\_HASH\_MODE\_AUTH mode

typedef struct **\_CpaCySymHashSetupData CpaCySymHashSetupData**

Hash Setup Data.

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

typedef enum **\_CpaCySymAlgChainOrder CpaCySymAlgChainOrder**

Algorithm Chaining Operation Ordering.

This enum defines the ordering of operations for algorithm chaining.

typedef struct **\_CpaCySymSessionSetupData CpaCySymSessionSetupData**

Session Setup Data.

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

typedef struct **\_CpaCySymOpData CpaCySymOpData**

Cryptographic Component Operation Data.

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

### See also:

**CpaCySymPacketType**

### Note:

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

typedef struct **\_CpaCySymStats CpaCySymStats**

## 14.7 Typedef Documentation

### Cryptographic Component Statistics.

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

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

Definition of callback function.

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

#### Context:

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

#### Assumptions:

None

#### Side-Effects:

None

#### Reentrant:

No

#### Thread-safe:

Yes

#### Parameters:

[in] <i>pCallbackTag</i>	Opaque value provided by user while making individual function call.
[in] <i>status</i>	Status of the operation. Valid values are CPA_STATUS_SUCCESS and CPA_STATUS_FAIL.
[in] <i>operationType</i>	Identifies the operation type that was requested in the cpaCySymPerformOp function.
[in] <i>pOpData</i>	Pointer to structure with input parameters.
[in] <i>pDstBuffer</i>	Caller MUST allocate a sufficiently sized destination buffer to hold the data output. For out-of-place processing the data outside the crypto regions in the source buffer are copied into the destination buffer. To perform "in-place" processing set the pDstBuffer parameter in cpaCySymPerformOp function to point at the same location as pSrcBuffer. For optimum performance, the data pointed to SHOULD be 8-byte aligned.
[in] <i>verifyResult</i>	This parameter is valid when the digestVerify option is set in the CpaCySymOpData structure. A value of CPA_TRUE indicates that the compare succeeded. A value of CPA_FALSE indicates that the compare failed for an unspecified reason.

#### Return values:

None

#### Precondition:

Component has been initialized.

#### Postcondition:

None

#### Note:



None

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


---

## 14.8 Enumeration Type Documentation

### enum **\_CpaCySymPacketType**

Packet type for the cpaCySymPerformOp function.

Enumeration which is used to indicate to the symmetric crypto perform function what type of packet the operation is required to be invoked on. The permitted types are a full packet, a partial packet, or the last part of a multi-part partial packet. 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:**

Partial packet processing is only supported for in-place cipher or in-place hash or in-place authentication operations. It does not apply to hash mode nested or algorithm chaining operations. The term "in-place operations" means that the result of the cipher or hash is written back into the source buffer.

**See also:****cpaCySymPerformOp()****Enumerator:**

<i>CPA_CY_SYM_PACKET_TYPE_FULL</i>	Perform an operation on a full packet.
<i>CPA_CY_SYM_PACKET_TYPE_PARTIAL</i>	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.
<i>CPA_CY_SYM_PACKET_TYPE_LAST_PARTIAL</i>	Complete the last part of a multi-part operation.

### enum **\_CpaCySymOp**

Types of operations supported by the cpaCySymPerformOp function.

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

**See also:****cpaCySymPerformOp****Enumerator:**

<i>CPA_CY_SYM_OP_CIPHER</i>	Cipher only operation on the data.
<i>CPA_CY_SYM_OP_HASH</i>	Hash only operation on the data.
<i>CPA_CY_SYM_OP_ALGORITHM_CHAINING</i>	Chain any cipher with any hash operation.
	The order depends on the value in the CpaCySymAlgChainOrder enum.

### enum **\_CpaCySymCipherAlgorithm**

## 14.8 Enumeration Type Documentation

Cipher algorithms.

This enumeration lists supported cipher algorithms and modes.

### Enumerator:

<i>CPA_CY_SYM_CIPHER_NULL</i>	NULL cipher algorithm.
	No mode applies to the NULL algorithm.
<i>CPA_CY_SYM_CIPHER_ARC4</i>	(A)RC4 cipher algorithm
<i>CPA_CY_SYM_CIPHER_AES_ECB</i>	AES algorithm in ECB mode.
<i>CPA_CY_SYM_CIPHER_AES_CBC</i>	AES algorithm in CBC mode.
<i>CPA_CY_SYM_CIPHER_AES_CTR</i>	AES algorithm in Counter mode.
<i>CPA_CY_SYM_CIPHER_AES_CCM</i>	AES algorithm in CCM mode.
	This authenticated cipher is only supported when the hash mode is also set to <i>CPA_CY_SYM_HASH_MODE_AUTH</i> . When this cipher algorithm is used the <i>CPA_CY_SYM_HASH_AES_CCM</i> element of the <i>CpaCySymHashAlgorithm</i> enum MUST be used to set up the related <i>CpaCySymHashSetupData</i> structure in the session context.
<i>CPA_CY_SYM_CIPHER_AES_GCM</i>	AES algorithm in GCM mode.
	This authenticated cipher is only supported when the hash mode is also set to <i>CPA_CY_SYM_HASH_MODE_AUTH</i> . When this cipher algorithm is used the <i>CPA_CY_SYM_HASH_AES_GCM</i> element of the <i>CpaCySymHashAlgorithm</i> enum MUST be used to set up the related <i>CpaCySymHashSetupData</i> structure in the session context.
<i>CPA_CY_SYM_CIPHER_DES_ECB</i>	DES algorithm in ECB mode.
<i>CPA_CY_SYM_CIPHER_DES_CBC</i>	DES algorithm in CBC mode.
<i>CPA_CY_SYM_CIPHER_3DES_ECB</i>	Triple DES algorithm in ECB mode.
<i>CPA_CY_SYM_CIPHER_3DES_CBC</i>	Triple DES algorithm in CBC mode.
<i>CPA_CY_SYM_CIPHER_3DES_CTR</i>	Triple DES algorithm in CTR mode.

### enum **\_CpaCySymCipherDirection**

Symmetric Cipher Direction.

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

### Enumerator:

<i>CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT</i>	Encrypt Data.
<i>CPA_CY_SYM_CIPHER_DIRECTION_DECRYPT</i>	Decrypt Data.

### enum **\_CpaCySymHashMode**

Symmetric Hash mode.

This enum indicates the Hash Mode.

### Enumerator:

<i>CPA_CY_SYM_HASH_MODE_PLAIN</i>	Plain hash.
-----------------------------------	-------------

## 14.8 Enumeration Type Documentation

<i>CPA_CY_SYM_HASH_MODE_AUTH</i>	Authenticated hash - HMAC & AES_XCBC_MAC algorithms.
	This mode MUST also be set to make use of the AES GCM and AES CCM algorithms.
<i>CPA_CY_SYM_HASH_MODE_NESTED</i>	Nested hash.

### enum **\_CpaCySymHashAlgorithm**

Hash algorithms.

This enumeration lists supported hash algorithms.

#### Enumerator:

<i>CPA_CY_SYM_HASH_MD5</i>	MD5 algorithm.
<i>CPA_CY_SYM_HASH_SHA1</i>	Supported in all 3 hash modes 128 bit SHA algorithm.
<i>CPA_CY_SYM_HASH_SHA224</i>	Supported in all 3 hash modes 224 bit SHA algorithm.
<i>CPA_CY_SYM_HASH_SHA256</i>	Supported in all 3 hash modes 256 bit SHA algorithm.
<i>CPA_CY_SYM_HASH_SHA384</i>	Supported in all 3 hash modes 384 bit SHA algorithm.
<i>CPA_CY_SYM_HASH_SHA512</i>	Supported in all 3 hash modes 512 bit SHA algorithm.
<i>CPA_CY_SYM_HASH_AES_XCBC</i>	Supported in all 3 hash modes AES XCBC algorithm.
<i>CPA_CY_SYM_HASH_AES_CCM</i>	This is only supported in the hash mode <i>CPA_CY_SYM_HASH_MODE_AUTH</i> . AES algorithm in CCM mode.
<i>CPA_CY_SYM_HASH_AES_GCM</i>	This authenticated cipher requires that the hash mode is set to <i>CPA_CY_SYM_HASH_MODE_AUTH</i> . When this hash algorithm is used, the <i>CPA_CY_SYM_CIPHER_AES_CCM</i> element of the <i>CpaCySymCipherAlgorithm</i> enum MUST be used to set up the related <i>CpaCySymCipherSetupData</i> structure in the session context. AES algorithm in GCM mode.
	This authenticated cipher requires that the hash mode is set to <i>CPA_CY_SYM_HASH_MODE_AUTH</i> . When this hash algorithm is used, the <i>CPA_CY_SYM_CIPHER_AES_GCM</i> element of the <i>CpaCySymCipherAlgorithm</i> enum MUST be used to set up the related <i>CpaCySymCipherSetupData</i> structure in the session context.

### enum **\_CpaCySymAlgChainOrder**

Algorithm Chaining Operation Ordering.

14.9 Function Documentation

This enum defines the ordering of operations for algorithm chaining.

Enumerator:

*CPA\_CY\_SYM\_ALG\_CHAIN\_ORDER\_HASH\_THEN\_CIPHER* Perform the hash operation followed by the cipher operation.

If it is required that the result of the hash (i.e. The digest) is going to be included in the data to be ciphered, then: a) 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. b) 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. c) The messageLenToCipherInBytes must be a multiple to the block size if a block cipher is being used.



*CPA\_CY\_SYM\_ALG\_CHAIN\_ORDER\_CIPHER\_THEN\_HASH* Perform the cipher operation followed by the hash operation, i.e.

The hash operation will work on the cipher text result of the cipher operation

14.9 Function Documentation

<b>CpaStatus</b> cpaCySymSessionCtxGetSize (	const <b>CpaInstanceHandle</b>	<i>instanceHandle,</i>
	const	
	<b>CpaCySymSessionSetupData</b>	<i>pSessionSetupData,</i>
	*	
	<b>Cpa32U</b> *	<i>pSessionCtxSizeInBytes</i>
)		

Cryptographic Symmetric Session Context Size Get Function.

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

## 14.9 Function Documentation

before the client calls the **cpaCySymInitSession** function.

### Context:

This is a synchronous function that can 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

### Parameters:

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

### Return values:

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

### Precondition:

The component has been initialized via **cpaCyStartInstance** function.

### Postcondition:

None

### Note:

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

### See also:

**CpaCySymSessionSetupData cpaCySymInitSession(), cpaCySymPerformOp()**

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

## 14.9 Function Documentation

### Cryptographic Component Symmetric Session Initialization Function.

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

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

No.

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

[in]	<i>instanceHandle</i>	Instance handle.
[in]	<i>pSymCb</i>	Pointer to callback function to be registered. Set to NULL if the cpaCySymPerformOp function is required to work in a synchronous manner.
[in]	<i>pSessionSetupData</i>	Pointer to session setup data which contains parameters which are static for a given cryptographic session such as operation type, mechanisms, and keys for cipher and/or hash operations.
[out]	<i>pSessionCtx</i>	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:**

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resources.

**Precondition:**

The component has been initialized via cpaCyStartInstance function.

**Postcondition:**

None

**Note:**

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

**See also:**

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

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

Cryptographic Component Symmetric Session Remove Function.

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 can 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

**Parameters:**

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

**Return values:**

**CPA\_STATUS\_SUCCESS** Function executed successfully.  
**CPA\_STATUS\_FAIL** Function failed.  
**CPA\_STATUS\_RETRY** Resubmit the request.  
**CPA\_STATUS\_INVALID\_PARAM** Invalid parameter passed in.  
**CPA\_STATUS\_RESOURCE** Error related to system resources.

**Precondition:**

The component has been initialized via **cpaCyStartInstance** function.

**Postcondition:**

None

**Note:**

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

**See also:**

**CpaCySymSessionCtx**, **cpaCySymInitSession()**

```

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

```

Cryptographic Component Symmetric Operation Perform Function.

Performs a cipher, hash or combined (cipher and hash) operation on the source data buffer using supported symmetric key algorithms and modes. This function maintains cryptographic state between calls for the partial crypto operations. If a partial crypto operation is being performed, then on a per-session basis, the next part of the multi-part message can be submitted prior to previous parts being completed, the only limitation being that all parts must be performed in sequential order. If for any reason a client wishes to terminate the partial packet processing on the session (for example if a packet fragment was lost) then the client MUST remove the session.

When 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.

Partial packet processing is only supported for in-place cipher or in-place hash operations. It does not apply to nested hash mode or algorithm chaining. The data on which the partial packet operation is to be performed MUST NOT be chained. There MUST be sufficient space in the buffer to store the result of the partial packet operation.

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

**Context:**

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

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

Yes when configured to operate in synchronous mode.

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

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



		Pointer to a structure containing request parameters. The client code allocates the memory for this structure. This component takes ownership of the memory until it is returned in the callback.
[in]	<i>pSrcBuffer</i>	Caller MUST allocate source buffer and populate 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.
[out]	<i>pDstBuffer</i>	Caller MUST allocate a sufficiently sized destination buffer to hold the data output. For out-of-place processing the data outside the crypto 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.
[out]	<i>pVerifyResult</i>	In synchronous mode, this parameter is returned when the digestVerify option is set in the CpaCySymOpData 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:**

<i>CPA_STATUS_SUCCESS</i>	Function executed successfully.
<i>CPA_STATUS_FAIL</i>	Function failed.
<i>CPA_STATUS_RETRY</i>	Resubmit the request.
<i>CPA_STATUS_INVALID_PARAM</i>	Invalid parameter passed in.
<i>CPA_STATUS_RESOURCE</i>	Error related to system resource.

**Precondition:**

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

**Postcondition:**

None

**Note:**

When in asynchronous mode, a callback of type CpaCySymCbFunc is generated in response to this function call. Any errors generated during processing are reported as part of the callback status code. Partial packet processing is only supported for in-place cipher or in-place hash operations (meaning when the output of the cipher or hash is written back into the source buffer).

**See also:**

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

```
CpaStatus cpaCySymQueryStats ( const CpaInstanceHandle instanceHandle,
                             CpaCySymStats * pSymStats
                             )
```

Query symmetric cryptographic statistics for a specific instance.

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

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

**Context:**

## 14.9 Function Documentation

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

**Assumptions:**

None

**Side-Effects:**

None

**Blocking:**

Yes

**Reentrant:**

No

**Thread-safe:**

Yes

**Parameters:**

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

**Return values:**

*CPA\_STATUS\_SUCCESS* Function executed successfully.  
*CPA\_STATUS\_FAIL* Function failed.  
*CPA\_STATUS\_INVALID\_PARAM* Invalid parameter passed in.

**Precondition:**

Component has been initialized.

**Postcondition:**

None

**Note:**

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

**See also:**

**CpaCySymStats**