**Virtio-PDCP Security Accelerator**

**g-API**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Author** | **Reason** |
| 01/22/2015 | 1 | NXP | Separated gAPI / Virtio sections from design document. This document details the g-API .  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Table of Contents 3](#_Toc441234777)

[1 Introduction 4](#_Toc441234778)

[2 References 5](#_Toc441234779)

[3 Abbreviations 5](#_Toc441234780)

[4 Scope 5](#_Toc441234781)

[5 PDCP Packet Processing – Look Aside Accelerator Packet Flow 6](#_Toc441234782)

[6 Application Usage 8](#_Toc441234783)

[7 g-APIs 10](#_Toc441234784)

[7.1 Accelerator Management APIs 10](#_Toc441234785)

[7.2 Functional APIs 10](#_Toc441234786)

[7.2.1 Control or setup APIs 10](#_Toc441234787)

[7.2.2 Data Processing APIs 10](#_Toc441234788)

[7.3 Accelerator Management APIs - definitions 11](#_Toc441234789)

[7.3.1 g\_pdcp\_la\_get\_api\_version 11](#_Toc441234790)

[7.3.2 g\_pdcp\_la\_open 11](#_Toc441234791)

[7.3.3 g\_pdcp\_la\_close 11](#_Toc441234792)

[7.4 Control or setup APIs – definitions 12](#_Toc441234793)

[7.4.1 g\_pdcp\_la\_capabilities\_get 12](#_Toc441234794)

[7.4.2 g\_pdcp\_la\_notification\_hooks\_register 12](#_Toc441234795)

[7.4.3 g\_pdcp\_la\_notification\_hooks\_deregister 12](#_Toc441234796)

[7.4.4 g\_pdcp\_la\_sec\_create 13](#_Toc441234797)

[7.4.5 g\_pdcp\_la\_sec\_get 14](#_Toc441234798)

[7.4.6 g\_pdcp\_la\_sec\_del 15](#_Toc441234799)

[7.5 Data Processing APIs – definitions 16](#_Toc441234800)

[7.5.1 g\_pdcp\_la\_process\_packet 16](#_Toc441234801)

[7.5.2 g\_pdcp\_la\_get\_maxq 17](#_Toc441234802)

[7.5.3 g\_pdcp\_la\_get\_queue 17](#_Toc441234803)

[7.5.4 g\_pdcp\_la\_poll 17](#_Toc441234804)

[7.6 G-API data types 18](#_Toc441234805)

# Introduction

A recent uplink/downlink performance analysis of an LTE protocol stack on a representative virtual mobile platform [[2](http://www.hindawi.com/journals/vlsi/2013/369627/#B2), [3](http://www.hindawi.com/journals/vlsi/2013/369627/#B3)] has identified the Protocol Data Convergence Protocol (PDCP) as the most time-critical component within the Layer 2 software architecture. PDCP incorporates two computationally expensive tasks:

-The ciphering and/or integrity algorithms, responsible for user data protection and for providing a secure communication.

-The Robust Header Compression (ROHC) algorithm, which compresses IP packet headers.

While both protocol functions show long processing times, ciphering & integrity comes in the first place followed by ROHC.

CPU cycles consumed in software based PDCP Security processing envolves:

1. IV construction: Concatenation of negotiated key, HFN|SN, Direction and bearer.
2. Applying crypto algorithm on the PDU using IV computed above.
3. To perform confidentiality and integrity protection requires above 2 steps repeated.

With accelerators offering PDCP Protocol offload can save CPU cycles that makes it worth by boosting performance.

Some of network processors have H/W accelerators supporting PDCP Security acceleration, that can be leveraged to offload the most computationally extensive task i.e. security algorithms for data confidentiality and integrity protection.

On the other hand, virtualization of logically separated layers of LTE stack are in great demands by most of the service providers for well-known advantages of virtualized software.

Given that PDCP security accelerator is available for offload on the host processor, for PDCP running as VNF under guest Virtual Machine (VM), we have a challenge to leverage the accelerator on host processor from VM.

In this document, we are proposing a virtio-pci based PDCP Security driver and device as per the Virtio Standards, so that the VNF can use the Virtio PDCP Security driver to access the PDCP Security Accelerator available on host processor. Using the standard Virtio-pci based Driver the VNF can access any underlying vendor specific PDCP Security Accelerator.

# References

[1] [The 3rd Generation Partnership Project (3GPP)](http://www.3gpp.org)

[2] D. Szczesny, S. Hessel, A. Showk, A. Bilgic, U. Hildebrand, and V. Frascolla, “Performance analysis of LTE protocol processing on an ARM based mobile platform,” in Proceedings of the 11th International Symposium on System-on-Chip (SoC '09), pp. 56–63, Tampere, Finland, October 2009. [View at Publisher](http://dx.doi.org/10.1109/SOCC.2009.5335678)· [View at Google Scholar](http://scholar.google.com/scholar?q=http://dx.doi.org/10.1109/SOCC.2009.5335678) · [View at Scopus](http://www.scopus.com/scopus/inward/record.url?eid=2-s2.0-74549138321&partnerID=K84CvKBR&rel=3.0.0&md5=adcb924743fb3756a8a46ebe2befd992)

[3] D. Szczesny, S. Hessel, A. Showk, A. Bilgic, U. Hildebrand, and V. Frascolla, “Joint uplink and downlink performance profiling of LTE protocol processing on a mobile platform,” International Journal of Embedded and Real-Time Communication Systems, vol. 1, no. 4, pp. 21–39, 2010. [View at Publisher](http://dx.doi.org/10.4018/jertcs.2010100102) ·[View at Google Scholar](http://scholar.google.com/scholar_lookup?title=Joint+uplink+and+downlink+performance+profiling+of+LTE+protocol+processing+on+a+mobile+platform&author=D.+Szczesny&author=S.+Hessel&author=A.+Showk&author=A.+Bilgic&author=U.+Hildebr&author=V.+Frascolla&publication_year=2010)

[4] [Packet Data Convergence Protocol (PDCP ... - 3GPP)](http://www.3gpp.org/dynareport/36323.htm)

# Abbreviations

VNF : Virtual Network Function

PDCP : Packet Data Convergence Protocol.

SRB : Signaling Radio Bearer.

DRB : Data Radio Bearer.

VM : Virtual Machine

KVM : Kernel Virtual Machine

MAC-I : Message Authentication Code – Integrity

SN : Sequence Number

PCI : Peripheral Component Interconnect

DL : Downlink

UL : Uplink

LCM : Least Common Multiple

# Scope

This document identifies a Virtio PDCP Security Accelerator which will perform PDCP Record Layer Acceleration. (e.g. Freescale SEC engine). In this case, the Guest VM can push PDCP Security Context into the hardware accelerator. Subsequently when buffers are submitted, the accelerator can perform Encap Security processing (clear packet to encrypted packet and/or add MAC-I for RBs) or Decap Security processing (encrypted packet to clear packet and/or authenticate for RBs) as required. This belongs to Look Aside class of accelerators as, the Guest VM submits packets to the accelerator and receives the processed packet from the accelerator before sending the packet out.

# PDCP Packet Processing – Look Aside Accelerator Packet Flow

 Host User Space

Hardware

VNF

Virtio-net Frontend

Virtio-net Frontend

Virtio-pdcp Frontend

Guest User Space

ODP based

PDCP

QEMU

vRING

 Transport

Virtio-pdcp Backend

Host Kernel

KVM

vHost-net

PDCP Security Accelerator

NIC

Backplane Processing RLC/MAC/PHY

Backhaul Processing GTPU/IPSec/QoS

EPC

Guest Kernel Space



Figure 1 PDCP Packet Processing –Look Aside Accelerator Flow

Figure 1 shows the flow of packets when PDCP Look aside accelerator is used. This figure shows

Packet flow between EPC – eNodeB - RLC-MAC:

* Packets processed by Vhost-Net
* Packet announced to VNF through Virtio-Net driver
* Packets arrive at the PDCP module for Processing
* As packets are submitted by the PDCP Module to the Virtio-pdcp front end driver, the buffers are put in the Virtio Descriptor Vrings or Virt Qs to be transferred to the Virtio-pdcp Backend.
* The Virtio-pdcp Backend is responsible for translating the packets from Virt Q Descriptor to the actual hardware accelerator in a message that the accelerator understands and vice-versa.
* The Virtio-pdcp Backend is also responsible for picking up processed packets from the hardware accelerator, updating the VirtQ rings and notifying the Guest VNF.
* The processed packets are sent out through the Virtio-net interface for post security processing.

# Application Usage

**virtio\_pdcp\_process\_cbk()** Receive processed native in/out buffer and it’s count, status.

Poll on specific Virtqueue to receive packets on registered callback function. g\_virtio\_pdcp\_poll(vq1)

**PDCP Application**

Pass security parameters, application handle and packet processing cbk function.

Store the pdcp context handle.

 Send native in/out buffer and it’s count

 Pass pdcp context handle

 to delete.

Check status (success/failure\_

In buf

Out buf

**g\_virtio\_pdcp\_poll**

Poll on selected Virtqueue’s

Used vring

**g\_pdcp\_sec\_create()** – Create virtio\_pdcp\_context\_block, fills the security params & Command type (VIRTIO\_PDCP\_CTRL\_ADD\_SEC\_CONTEXT) into PDCPhdr (virtio\_net\_hdr\_mz). Enqueue to avail vring and notify. call g\_virtio\_pdcp\_poll() & wait for the response

**Frontend** [Guest]

]

**g\_pdcp\_process\_packet()** – Make in/out buffers as single chain of buffers, **VIRTIO\_PDCP\_CTRL\_PROCESS\_PKT**.

 Enqueue to avail vring and notify.

Return to caller,unblocking call.

**g\_pdcp\_sec\_del()** –Pass the backend handle,command type (**VIRTIO\_PDCP\_CTRL\_DEL\_SEC\_CONTEXT)**

Avail vRing

Used vRing

 **Virtqueue - 1**

**Data Virtqeue (s)**

Parse Security params

**VIRTIO\_PDCP\_CTRL\_ADD\_SEC\_CONTEXT** – Create security context, return the backend handle and the status.

Parse the Header for commands

**Backend** [Host]

**VIRTIO\_PDCP\_CTRL\_PROCESS\_PKT** – Find the UL/DL sec context. Apply security, send processed out buffers as well as in buffers and status.

Fetch the backend handle

 **VIRTIO\_PDCP\_CTRL\_DEL\_SEC\_CONTEXT** – Delete the security context. Return the status.

Parse native input buffers

Figure 2: G-API functions

Refer Figure 2 for the G-API functional flows.

All the command G-API (CREATE/READ/DELETE) supports both SYNC/ASYNC modes of operation.

If **G\_PDCP\_CTRL\_FLAG\_ASYNC** flag is set, API call be asynchronous. Otherwise, API call will be synchronous.

# g-APIs

The application Interface APIs (g-APIs) have two components, namely the Accelerator Management APIs and the functional APIs.

## Accelerator Management APIs

The following APIs shall be supported for Accelerator Management.

1. g\_pdcp\_la\_get\_api\_version
2. g\_pdcp\_la\_open
3. g\_pdcp\_la\_close

## Functional APIs

The functional APIs are in turn classified to control or setup APIs and data processing APIs. Each API requires an accelerator handle, which the application must have obtained by calling g\_pdcp\_la\_open() function.

## 7.2.1 Control or setup APIs

1. g\_pdcp\_la\_capabilities\_get
2. g\_pdcp\_la\_notification\_hooks\_register
3. g\_pdcp\_la\_notification\_hooks\_deregister
4. g\_pdcp\_la\_sec\_create
5. g\_pdcp\_la\_sec\_get
6. g\_pdcp\_la\_sec\_del

## 7.2.2 Data Processing APIs

1. g\_pdcp\_la\_process\_packet
2. g\_pdcp\_la\_get\_maxq
3. g\_pdcp\_la\_get\_queue
4. g\_pdcp\_la\_poll

## Accelerator Management APIs - definitions

### g\_pdcp\_la\_get\_api\_version

int32\_t g\_pdcp\_la\_get\_api\_version(char \*version);

/\* Function Name: g\_pdcp\_la\_get\_api\_version

\* Input/Output: a variable to hold the version

\* Return Value: G\_PDCP\_SUCCESS (or) G\_PDCP\_FAILURE

\* Description: Application to use this api to get the API version \*/

Application can use this API to get the underlying API version.

### g\_pdcp\_la\_open

int32\_t g\_pdcp\_la\_open(

struct g\_pdcp\_la\_open\_inargs \*in,

struct g\_pdcp\_la\_open\_outargs \*out);

/\* Function Name: g\_pdcp\_la\_open

\* in: application identity, callback function to invoke when the underlying accelerator connection is broken, callback argument and length of the same.

out: handle to the accelerator

\* Return Value: G\_PDCP\_SUCCESS (or) G\_PDCP\_FAILURE

\* Description : Get a handle to an PDCP Look Aside Accelerator Instance.

\*/

An Application shall use this API to open a virtual The application registers a callback function to be invoked, if the underlying virtual accelerator association is broken. The application is expected to take corrective action such as closing the current handle and opening a new handle if required.

### g\_pdcp\_la\_close

int32\_t g\_pdcp\_la\_close(struct g\_pdcp\_la\_handle \*handle);

/\* Function Name: g\_pdcp\_la\_close

\* Input : g\_pdcp\_la\_handle \*handle

\* Output : None

\* Return Value: G\_PDCP\_SUCCESS (or) G\_PDCP\_FAILURE

\* Description : Given a handle, close the virtual accelerator instance \*/

Application should use this API to close the handle of the previously opened accelerator instance. Application may no longer access the underlying accelerator.

## Control or setup APIs – definitions

### g\_pdcp\_la\_capabilities\_get

int32\_t g\_pdcp\_la\_capabilities\_get(

struct g\_pdcp\_la\_handle \*handle,

enum g\_pdcp\_la\_control\_flags flags,

struct g\_pdcp\_la\_cap\_get\_outargs \*out,

struct g\_pdcp\_la\_resp\_args \*resp);

/\* Function Name: g\_pdcp\_la\_capabilities\_get

\* Input: handle – accelerator handle with optional group handle; flags indicating SYNC or ASYNC, Response required or not; In this case response is required.

out – Pointer to the output parameter structure (Capabilities);

resp – Response callback function and details in case ASYNC response is requested

 \* Output: Success or Failure

 \* Description: Returns the capabilities of the underlying accelerator.

In the case of synchronous response, the out parameter has the capabilities, otherwise, the resp callback function is invoked with the capabilities

\*/

Application can call this API to find out the capabilities offered by the underlying virtual PDCP accelerator. The response may be returned synchronously or asynchronously based on the Application’s preference as set by the flags argument. When returned synchronously, the capabilities are returned by the out parameter.

When returned asynchronously, the capabilities are passed as type struct g\_pdcp\_la\_cap\_get\_outargs through the response callback function.

### g\_pdcp\_la\_notification\_hooks\_register

 TBD

### g\_pdcp\_la\_notification\_hooks\_deregister

 TBD

### g\_pdcp\_la\_sec\_create

int32\_t g\_pdcp\_la\_sec\_create(

 const struct g\_pdcp\_sec\_la\_add\_inargs \*in,

 struct g\_pdcp\_sec\_la\_add\_outargs \*out,

 struct g\_pdcp\_la\_resp\_args \*resp);

/\* Function Name: g\_pdcp\_la\_sec\_create

 \*Parameters:

 \*handle: Accelerator handle,

 \*Input Arguments: in {flags,sec\_params, pkt\_cb\_fn, add\_cb\_fn, ctxt\_handle}

 \* flags: Synchronous or asynchronous, Response required or not;

 \* sec\_params: security parameters,

 \* pkt\_cb\_fn: packet processing call back function

\* add\_cb\_fn: call back function when the response is expected asynchronously

 \* Out Argument: out - Result and Security Handle;

 \* resp: Response callback

 \* function and callback argument in case ASYNC response is

 \* requested

 \* Return Value: G\_PDCP\_SUCCESS (or) G\_PDCP\_FAILURE

 \* Description: Application uses this API to create PDCP Security

 \* context

\*/

Application can call this API to create PDCP Security context. This API returns G\_PDCP\_SUCCESS when the context has been successfully (if synchronous response is expected) created by the Virtual Accelerator. A Handle is returned by this API in the response. Application is expected to use this Handle in subsequent calls such as g\_pdcp\_la\_sec\_del , or one of the Read context commands (g\_pdcp\_la\_sec\_get ) or packet processing commands (g\_pdcp\_la\_process\_packet).

This initiates VIRTIO\_PDCP\_CTRL\_ADD\_SEC\_CONTEXT command to backend driver.

The response can be synchronous or asynchronous depending on the flags in input arguments. After enqueue PDCP driver polls for the status message from the Host, updates the backend handle in corresponding context maintained by Frontend driver and return to the application.

### g\_pdcp\_la\_sec\_get

 int32\_t g\_pdcp\_la\_sec\_get(

 struct g\_pdcp\_la\_handle \*handle,

 const struct g\_pdcp\_la\_sec\_get\_inargs \*in,

 struct g\_pdcp\_la\_sec\_get\_outargs \*out,

 struct g\_pdcp\_la\_resp\_args \*resp);

/\* Function Name: g\_pdcp\_la\_sec\_get

 \* Input: Virtual Accelerator Handle (handle/group handle), Input

 \* arguments that includes sec\_handle (valid for get exact or

 \* get next calls) Operation Get First/Get First N/

 \* Get Next/Get Next N/Get Exact/, number of

 \* contexts to read

 \* flags: API control flags, out: contains required

 \* memory to hold the output information,

 \* result: SUCCESS or error code; resp: Optional response

 \* callback function and arguments, in case ASYNC flag is set.

 \* Return Value: Success or Error

 \* Description: Application/Sub-application can call this API to read

 \* Security Information or statistics

 \*/

Application can use this API to retrieve security context or statistics. For convenience several flags are available, such as ‘get first’, get first n number of contexts, get next, get next n number of contexts and get\_exact. Application has the flexibility to get either the security information or the statistics.

This is synchronous call. After enqueue PDCP driver polls for the status message from the Host and return to the application.

### g\_pdcp\_la\_sec\_del

 int32\_t g\_pdcp\_la\_sec\_del(struct g\_pdcp\_la\_handle \*handle,

 const struct g\_pdcp\_la\_sec\_del\_inargs \*in,

 struct g\_pdcp\_la\_sec\_del\_outargs \*out,

 struct g\_pdcp\_la\_resp\_args \*resp);

/\* Function Name: g\_pdcp\_la\_sec\_del

 \* Input: Accelerator Handle, Security context Handle

 \* Input/Output: Success or error code

 \* Description: Given the virtual accelerator handle and security handle, delete the context

 \*/

Application calls this API to delete the PDCP security context the virtual accelerator. This initiates VIRTIO\_PDCP\_CTRL\_ DEL\_SEC\_CONTEXT command to backend driver.

This is synchronous call. After enqueue PDCP driver polls for the status message from the Host and return to the application.

## Data Processing APIs – definitions

### g\_pdcp\_la\_process\_packet

 int32\_t g\_pdcp\_la\_process\_packet(

struct g\_pdcp\_la\_handle \*handle,

struct g\_pdcp\_la\_process\_packet\_inargs \*in,

 struct g\_pdcp\_la\_resp\_args \*resp);

/\*

 \* Function Name: g\_pdcp\_la\_process\_packet

 \* Input: Accelerator handle,

 \* Input Arguments (

 \* handle – security handle

 \* n\_in\_bufs – no.of in buffers,

 \* n\_out\_bufs – no.of out buffers

 \* pdcp\_count

 \* \*inbuf – Chain of input buffers pointer

 \* \*outbuf - Chain of output buffers pointer

 \* \*pkt\_opaque – opaque pointer

 \* return :result

 \* Success or error code to indicate packet has been submitted

 \* to accelerator or not. Resp: includes the callback function

 \* that will be called on completion of packet processing.

 \*/

Application calls this API for PDCP security processing on the packet. When the application submits the security Handle, and the set of input and output(in resp args) buffers to the virtual accelerator, the application expects the virtual accelerator for PDCP security processing.

The response can be synchronous or asynchronous depending on the flags in input arguments. After enqueue PDCP driver polls for the status message from the Host and return to the application.

If asynchronous response is expected, after enqueue PDCP driver returns the status to the application. Application is expected to poll on the Queue by calling **g\_pdcp\_la\_poll** function to receive the processed buffers using the registered callback function during the g\_pdcp\_la\_sec\_create.

### g\_pdcp\_la\_get\_maxq

uint16\_t **g\_pdcp\_la\_get\_maxq**(void);

This function returns the maximum queue supported by the PDCP device.

### g\_pdcp\_la\_get\_queue

void **g\_pdcp\_la\_get\_queue**(uint16\_t queue\_id, void \*\*qptr);

This function returns the virtqueue pointer to the qptrs, which corresponds to queue\_id (queue index).

### g\_pdcp\_la\_poll

void **g\_pdcp\_la\_poll**(void \*vq)

Application calls this API for polling on the used ring. On reception of the packets, the registered packet processing callback invoked. This is non-blocking call, it checks the used ring of the given virtio queue, and if some job is available in the used ring it invokes the corresponding call back function to pass the information back to application. If the response is for create command, it updates the backend handle in corresponding context maintained by Frontend driver.

## G-API data types

#define CIPHER\_KEY\_LEN 16

#define AUTH\_KEY\_LEN 16

enum g\_pdcp\_auth\_alg {

 G\_PDCP\_AUTH\_ALG\_SNOWF9 = 1, /\* MD5 HMAC Authentication Algo. \*/

 G\_PDCP\_AUTH\_ALG\_AES = 2, /\* SHA1 HMAC Authentication Algo. \*

 G\_PDCP\_AUTH\_ALG\_ZUC = 3 , /\* AES-XCBC Authentication Algo. \*/

 G\_PDCP\_AUTH\_ALG\_NONE = 4, /\* No Authentication \*/

};

enum g\_pdcp\_cipher\_alg {

 G\_PDCP\_CIPHER\_ALG\_SNOWF8 = 1,

 G\_PDCP\_CIPHER\_ALG\_AES = 2,

 G\_PDCP\_CIPHER\_ALG\_ZUC = 3,

 G\_PDCP\_CIPHER\_ALG\_NULL = 4 /\* NULL Encryption algorithm \*/

};

enum g\_pdcp\_sn\_size

{

 G\_PDCP\_SN\_SIZE\_5=5,

 G\_PDCP\_SN\_SIZE\_7=7,

 G\_PDCP\_SN\_SIZE\_12=12,

 G\_PDCP\_SN\_SIZE\_15=15,

};

enum g\_pdcp\_proto\_dir

{

 G\_PDCP\_ENCAP = 1,

 G\_PDCP\_DECAP = 2

};

enum g\_pdcp\_pkt\_dir

{

 G\_PDCP\_UL = 1,

 G\_PDCP\_DL = 2

};

enum g\_pdcp\_notify\_type

{

G\_PDCP\_COUNT\_THRESHOLD = 1,

G\_PDCP\_REL\_COUNT\_THRESOLD = 2,

};

enum g\_pdcp\_la\_control\_flags

{

 G\_PDCP\_CTRL\_FLAG\_ASYNC = 1, /\* If Set, API call be asynchronous. Otherwise, API call will be synchronous \*/

 G\_PDCP\_CTRL\_FLAG\_NO\_RESP\_EXPECTED = 2, /\* If set, no response is expected for this API call \*/

};

Application shall use the above data structure to pass the response requested – async or sync and whether a response is required or not. This structure is a parameter in most of the APIs.

enum g\_pdcp\_sa\_get\_op {

 G\_PDCP\_LA\_SEC\_GET\_FIRST\_N = 1,

 G\_PDCP\_LA\_SEC\_GET\_NEXT\_N = 2,

 G\_PDCP\_LA\_SEC\_GET\_EXACT = 3

};

enum g\_pdcp\_return\_codes

{

 G\_PDCP\_SUCCESS= 1,

 G\_PDCP\_FAILURE = 2,

};

struct g\_pdcp\_la\_handle {

 void \*handle;

};

struct g\_pdcp\_sec\_handle {

 void \*handle; /\* context handle \*/

 };

struct g\_pdcp\_auth\_algo\_cap {

 uint32\_t snow\_f9:1,

 aes:1,

 zuc:1,

 none:1

};

struct g\_pdcp\_cipher\_algo\_cap {

 uint32\_t snowf8:1,

 aes:1,

 zuc:1,

 null:1;

};

struct g\_pdcp\_capabilities

{

 uint32\_t pdcp\_features; /\*subset of VIRTIO\_PDCP\_FEATURES negotiated with backend\*/

 struct g\_pdcp\_auth\_algo\_cap auth\_algo\_caps;

 struct g\_pdcp\_cipher\_algo\_cap cipher\_algo\_caps;

}

struct g\_pdcp\_la\_cap\_get\_outargs

{

 struct g\_pdcp\_capabilities caps; /\* Capabilities \*/

}

struct g\_pdcp\_sec\_params

{

 enum g\_pdcp\_auth\_alg auth\_algo;

 uint8\_t auth\_key[AUTH\_KEY\_LEN];

 uint32\_t auth\_key\_len\_bits;

 enum g\_pdcp\_cipher\_alg cipher\_algo;

 uint8\_t cipher\_key[CIPHER\_KEY\_LEN];

 uint32\_t cipher\_key\_len\_bits;

 uint8\_t g\_sn\_size;

 enum g\_pdcp\_pkt\_dir pkt\_dir;

 enum g\_pdcp\_proto\_dir proto\_dir;

 uint32\_t hfn;

 uint32\_t bearerid;

 uint32\_t count\_threshold;

}

typedef void(\***g\_pdcp\_add\_context\_resp\_cbfn**)(void \*cb\_arg, struct g\_pdcp\_sec\_la\_add\_outargs);

- The callback function is used to pass response for the PDCP create command (asynchronous mode) from Backend.

typedef void(\***g\_pdcp\_pkt\_process\_resp\_cbfn**)(void \*cb\_arg, void \*pkt\_handle,

 void \*inbuf, void \*outbuf,

 uint32\_t n\_in\_bufs, uint32\_t n\_out\_bufs,

 int32\_t result);

- The callback function is similar to the input arguments, it return the result, no. of input buffer, no. of output buffer, in and out chained buffer pointer.

struct g\_pdcp\_sec\_la\_add\_inargs

{

 enum g\_pdcp\_la\_control\_flags flags;

 struct g\_pdcp\_sec\_params sec\_params;

 void \*cb\_arg; /\*for asynchronous response\*/

 **g\_pdcp\_add\_context\_resp\_cbfn add\_cb\_fn;**

 g\_pdcp\_pkt\_process\_resp\_cbfn pkt\_cb\_fn;

 void \*ctxt\_handle;

};

struct g\_pdcp\_sec\_la\_add\_outargs {

 int32\_t result; /\* Non zero value: Success, Otherwise failure \*/

 struct g\_pdcp\_sec\_handle handle;

}

typedef void(\***g\_pdcp\_del\_context\_resp\_cbfn**)(void \*cb\_arg, struct g\_pdcp\_la\_sec\_del\_outargs

);

- The callback function is used to pass response for the PDCP delete command (asynchronous mode) from Backend.

struct g\_pdcp\_la\_sec\_del\_inargs

{

 enum g\_pdcp\_la\_control\_flags flags;

 struct g\_pdcp\_sec\_handle handle; /\* SA Handle \*/

 void \*cb\_arg; /\*For Asynchronous response\*/

 **g\_pdcp\_del\_context\_resp\_cbfn resp\_fn; /\*** For Asynchronous response **\*/**

};

struct g\_pdcp\_la\_sec\_del\_outargs

{

 int32\_t result;

};

struct g\_pdcp\_sec\_stats {

 uint64\_t packets\_processed; /\* Number of packets processed \*/

 uint64\_t bytes\_processed; /\* Number of bytes processed \*/

uint32\_t no\_tail\_room; /\* Number of packets with no tail room required for padding \*/

uint32\_t submit\_to\_accl\_failed; /\* Number of packets where submission to underlying hardware accelerator failed \*/

uint32\_t auth\_failures;

}

typedef void(\***g\_pdcp\_get\_sec\_resp\_cbfn**)(void \*cb\_arg, struct g\_pdcp\_la\_sec\_get\_outargs);

- The callback function is used to pass response for the PDCP GET command (asynchronous mode) from Backend.

struct g\_pdcp\_la\_sec\_get\_outargs {

{

 int32\_t result; /\* 0: Success: Non zero value: Error code indicating failure \*/

 int32\_t num\_out; /\*Number of output records\*/

 struct g\_pdcp\_sec\_patams \*params; /\* An array of params[] to hold ‘num\_out’ information \*/

 struct g\_pdcp\_sec\_stats \*stats; /\* An array of stats[] to hold the statistics \*/

 g\_pdcp\_sec\_handle \*handle; /\* handle returned to be used for subsequent Get Next N call \*/

};

struct g\_pdcp\_la\_sec\_get\_inargs

{

enum g\_pdcp\_la\_control\_flags flags;

 struct g\_pdcp\_sec\_handle \*handle; /\* Field is not applicable for get\_first \*/

 enum g\_pdcp\_la\_sec\_get\_op operation; /\* Get First, Next or Exact \*/

 struct g\_pdcp\_sec\_params sec\_params;

 void \*cb\_arg; /\*For Asynchronous response\*/

 **g\_pdcp\_get\_sec\_resp\_cbfn resp\_fn; /\*** For Asynchronous response **\*/**

 uint32\_t num; /\* Number of contexts to read \*/

 uint32\_t get\_flags; /\* flags indicate to get complete information or only Statistics \*/

}

struct g\_pdcp\_la\_process\_packet\_inargs {

 enum g\_pdcp\_la\_control\_flags flags;

 struct g\_pdcp\_sec\_handle handle; /\* SA Handle \*/

 uint32\_t n\_in\_bufs;

 uint32\_t n\_out\_bufs;

 uint32\_t pdcp\_count;

 void \*inbuf; /\* Chain of input buffers \*/

 void \*outbuf; /\* Chain of output buffers \*/

 void \*pkt\_opaque;

};

typedef void (\*g\_pdcp\_la\_instance\_broken\_cbk\_fn)(struct g\_pdcp\_la\_handle \*handle, void \*cb\_arg);

The above application registered callback function will be invoked, when underlying accelerator instance to which the handle is attached is removed.

struct g\_pdcp\_la\_open\_inargs {

uint16\_t pci\_vendor\_id;

uint16\_t device\_id;

char \*app\_identity; /\* Application identity \*/

g\_pdcp\_la\_instance\_broken\_cbk\_fn, /\* Callback function to be

called when the connection to the underlying accelerator is broken \*/

void \*cb\_arg; /\* Callback argument \*/

int32\_t cb\_arg\_len; /\* Callback argument length \*/

};