Virtio-IPSec Accelerator

g-API

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Author** | **Reason** |
| 07/07/2015 | 1 | Freescale Semiconductor | Initial version |
| 07/25/2015 | 2 | Freescale Semiconductor | Minor changes in data structures, naming |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Table of Contents 3](#_Toc424044050)

[**1** Introduction 6](#_Toc424044051)

[**2** References 6](#_Toc424044052)

[**3** Scope 6](#_Toc424044053)

[**4** IPsec Device Definition 6](#_Toc424044054)

[**5** System Overview (Virtio IPSec device) 7](#_Toc424044055)

[5.1 Lifecycle – Virtual Accelerator detection, programming and removal 8](#_Toc424044056)

[5.1.1 Discovery: 8](#_Toc424044057)

[5.1.2 Removal 9](#_Toc424044058)

[**6** Application Usage 10](#_Toc424044059)

[6.1.1 Modes 11](#_Toc424044060)

[6.1.2 Virtual Accelerator Assignment 11](#_Toc424044061)

[**7** g-APIs 12](#_Toc424044062)

[7.1 Accelerator Management APIs 12](#_Toc424044063)

[7.2 Functional APIs 12](#_Toc424044064)

[7.2.1 Control or setup APIs 12](#_Toc424044065)

[7.2.2 Data Processing APIs 12](#_Toc424044066)

[**8** g-API definitions 13](#_Toc424044067)

[8.1 g\_ipsec\_la\_get\_api\_version 13](#_Toc424044068)

[8.2 g\_ipsec\_la\_open 13](#_Toc424044069)

[8.3 g\_ipsec\_la\_create\_group 13](#_Toc424044070)

[8.4 g\_ipsec\_la\_delete\_group 14](#_Toc424044071)

[8.5 g\_ipsec\_la\_close 14](#_Toc424044072)

[8.6 g\_ipsec\_la\_available\_list\_get 15](#_Toc424044073)

[8.7 g\_ipsec\_la\_active\_list\_get 15](#_Toc424044074)

[8.8 g\_ipsec\_la\_get\_capabilities 15](#_Toc424044075)

[8.9 g\_ipsec\_la\_notification\_hooks\_register 16](#_Toc424044076)

[8.10 g\_ipsec\_la\_notifications\_hook\_deregister 16](#_Toc424044077)

[8.11 g\_ipsec\_la\_sa\_add 17](#_Toc424044078)

[8.12 g\_ipsec\_la\_sa\_mod 17](#_Toc424044079)

[8.13 g\_ipsec\_la\_sa\_del 18](#_Toc424044080)

[8.14 g\_ipsec\_la\_sa\_flush 18](#_Toc424044081)

[8.15 g\_ipsec\_la\_sa\_get 18](#_Toc424044082)

[8.16 g\_ipsec\_la\_packet\_encap 19](#_Toc424044083)

[8.17 g\_ipsec\_la\_packet\_decap 20](#_Toc424044084)

[8.18 g\_ipsec\_la\_multi\_packet\_encap 20](#_Toc424044085)

[8.19 g\_ipsec\_la\_multi\_packet\_decap 21](#_Toc424044086)

[**9** Data Structures 21](#_Toc424044087)

[9.1 g\_ipsec\_la\_create\_group\_inargs 21](#_Toc424044088)

[9.2 g\_ipsec\_la\_create\_group\_outargs 21](#_Toc424044089)

[9.3 g\_ipsec\_la\_instance\_broken\_cbk\_fn 22](#_Toc424044090)

[9.4 g\_ipsec\_la\_open\_inargs 22](#_Toc424044091)

[9.5 g\_ipsec\_la\_open \_outargs 22](#_Toc424044092)

[9.6 g\_ipsec\_la\_resp\_args 22](#_Toc424044093)

[9.7 g\_ipsec\_la\_handle 22](#_Toc424044094)

[9.8 g\_ipsec\_la\_sa\_handle 23](#_Toc424044095)

[9.9 g\_ipsec\_la\_auth\_algo\_cap 23](#_Toc424044096)

[9.10 g\_ipsec\_la\_cipher\_algo\_cap 23](#_Toc424044097)

[9.11 g\_ipsec\_la\_comb\_algo\_cap 23](#_Toc424044098)

[9.12 g\_ipsec\_la\_capabilities 23](#_Toc424044099)

[9.13 g\_ipsec\_cap\_get\_outargs 24](#_Toc424044100)

[9.14 g\_ipsec\_la\_resp\_cbfn 24](#_Toc424044101)

[9.15 g\_ipsec\_seq\_number\_notification 24](#_Toc424044102)

[9.16 g\_ipsec\_la\_cbk\_sa\_seq\_number\_overflow\_fn 24](#_Toc424044103)

[9.17 g\_ipsec\_la\_cbk\_sa\_seq\_number\_periodic\_update\_fn 25](#_Toc424044104)

[9.18 g\_ipsec\_la\_lifetime\_in\_bytes\_notification 25](#_Toc424044105)

[9.19 g\_ipsec\_la\_cbk\_sa\_soft\_lifetimeout\_expiry\_fn 25](#_Toc424044106)

[9.20 g\_ipsec\_la\_notification\_hooks 25](#_Toc424044107)

[9.21 g\_ipsec\_la\_sa\_crypto\_params 25](#_Toc424044108)

[9.22 g\_ipsec\_la\_ipcomp\_info 26](#_Toc424044109)

[9.23 g\_ipsec\_la\_tunnel\_end\_addr 26](#_Toc424044110)

[9.24 g\_ipsec\_la\_nat\_traversal\_info 26](#_Toc424044111)

[9.25 g\_ipsec\_la\_sa 26](#_Toc424044112)

[9.26 g\_ipsec\_la\_sa\_add\_inargs 27](#_Toc424044113)

[9.27 g\_ipsec\_la\_sa\_add\_outargs 27](#_Toc424044114)

[9.28 g\_ipsec\_la\_sa\_modify\_flags 27](#_Toc424044115)

[9.29 g\_ipsec\_la\_sa\_mod\_inargs 28](#_Toc424044116)

[9.30 g\_ipsec\_la\_sa\_modify\_outargs 28](#_Toc424044117)

[9.31 g\_ipsec\_la\_sa\_del\_inargs 28](#_Toc424044118)

[9.32 g\_ipsec\_la\_sa\_del\_outargs 28](#_Toc424044119)

[9.33 g\_ipsec\_la\_sa\_stats 29](#_Toc424044120)

[9.34 g\_ipsec\_la\_sa\_get\_outargs 29](#_Toc424044121)

[9.35 g\_ipsec\_la\_sa\_get\_inargs 30](#_Toc424044122)

[9.36 g\_ipsec\_la\_data 30](#_Toc424044123)

[9.37 g\_ipsec\_la\_packet 30](#_Toc424044124)

[**10** Macros 30](#_Toc424044125)

[**11** Enumerations 30](#_Toc424044126)

[11.1 g\_ipsec\_la\_mode 30](#_Toc424044127)

[11.2 g\_ipsec\_la\_control\_flags 31](#_Toc424044128)

[11.3 g\_ipsec\_auth\_alg 31](#_Toc424044129)

[11.4 g\_ipsec\_la\_cipher\_alg 31](#_Toc424044130)

[11.5 g\_ipsec\_la\_comb\_alg 31](#_Toc424044131)

[11.6 g\_ipsec\_la\_ipcomp\_alg 32](#_Toc424044132)

[11.7 g\_ipsec\_la\_dscp\_handle 32](#_Toc424044133)

[11.8 g\_ipsec\_la\_df\_handle 32](#_Toc424044134)

[11.9 g\_ipsec\_la\_sa\_direction 32](#_Toc424044135)

[11.10 g\_ipsec\_sa\_flags 32](#_Toc424044136)

[11.11 g\_ipsec\_la\_inb\_sa\_flags 32](#_Toc424044137)

[11.12 g\_ipsec\_la\_sa\_modify\_replay\_info\_flags 33](#_Toc424044138)

[11.13 g\_ipsec\_la\_sa\_get\_op 33](#_Toc424044139)

# Introduction

This document introduces g-APIs for IPsec. These APIs are defined that enable VNF applications can use to access the underlying IPsec h/w accelerator.

# References

Virtio Specifications

<http://docs.oasis-open.org/virtio/virtio/v1.0/virtio-v1.0.pdf>

<http://ozlabs.org/~rusty/virtio-spec/virtio-0.9.5.pdf>

Virtio-net, Vhost-net, Vhost-user implementations in Linux 3.19, Qemu 2.3.0

# Scope

This document identifies the necessary generic apis or g-apis for ipsec, that may be required for application to use to underlying ipsec accelerator. The g-apis cover the management and lifecycle APIs as well as the command and data processing APIs.

# IPsec Device Definition

A default IPsec Device is expected to provide the following functionality:

1. IPv4 Support
2. Tunnel and Transport Mode
3. ESP (Encapsulating Security Protocol)
4. Checksum to be calculated for Tunnel packets

An IPSec Device may exhibit other capabilities such as AH processing etc. Applications can learn about the same by invoking appropriate g-APIs.

# System Overview (Virtio IPSec device)

The Virtio IPsec device is emulated as a Virtio PCI based device. The high level picture of device and drivers as projected to the Guest is shown in Figure 1

Virtio-IPsec Frontend

Virtio Component

Qemu Device Emulation

Virtio-PCI IPsec device

Virtio IPsec device

PCI Bus

Virtio-PCI device driver

Virtio Bus

Virtio IPsec device driver

Application Interface Layer

g-API , Management

g-API driver Glue

Figure 1 Virtio IPsec Device and Driver

The Virtio IPsec Frontend driver contains two main components,

* Virtio Component
  + The Virtio Component interfaces with the underlying Virtio registering a driver to drive the Virtio IPsec device.
  + It comprises of two components, namely
    - Virtio PCI component
      * This is a Virtio Generic Module that acknowledges the Virtio Device before publishing the Virtio IPsec device on the Virtio Bus. (Part of existing code.)
    - Virtio Bus IPsec Component
      * This registers with the Virtio Bus so that the virtio-ipsec driver can drive the virtio ipsec device.
    - The driver provides an API interface that can be used by the g-API glue layer on behalf of the Application to communicate to the underlying accelerator
* Application Interface Component
  + The g-apis provide APIs for application and management layers to talk to the underlying accelerator. They include
    - Application interface
    - Management interface
  + The g-api glue layer will glue the g-apis to the underlying virtio-driver APIs.

## Lifecycle – Virtual Accelerator detection, programming and removal



Figure 2 Lifecycle - Virtio Device Assignment and Removal to VNF

Figure 2 shows the discovery and removal of Virtio-IPsec Look aside accelerator device.

### Discovery:

* Upon Qemu command line or similar invocation, a virtio-ipsec device is discovered on the Virtio-PCI bus
* The device is configured: Queues and Interrupt (virtio\_ipsec\_probe())
* The virtual accelerator device is added to the device list.

### Removal

* Upon Qemu command line or similar invocation, a virtio-ipsec device is removed
* The device is removed from the virtual IPsec instance list (virtio\_ipsec\_remove())

# Application Usage

VNF Applications may use an IPsec Accelerator instance in an exclusive mode or shared mode. In some cases a single application may make use of several accelerators

`

Application1

Application2

Virtio-IPsec-Instance 1

g\_ipsec\_la\_open()

Virtio-IPsec-Instance 2 g\_ipsec\_la\_open()

Application3

Virtio-IPsec-Instance 3

g\_ipsec\_la\_open()-App2

g\_ipsec\_la\_open()- App3

g\_ipsec\_la\_create\_group() – Sub-App1

g\_ipsec\_la\_create\_group() – Sub-App2

EXCLUSIVE

EXCLUSIVE

SHARED

Application 4

SHARED

EXCLUSIVE

Virtio-IPsec-Instance 4

g\_ipsec\_la\_open()

Application 5

Virtio-IPsec-Instance 5 g\_ipsec\_la\_create\_handle()

EXCLUSIVE

Figure 3 Application Usage Model

Figure 3 shows possible application models on how VNF may make use of Virtual accelerators. From a VNF perspective there may be several applications that may need to make use of the Virtual-IPsec Accelerator. Each application may have several groups as well. Here are examples for applications and groups

1. Linux IPsec supporting multiple namespaces. In this case, the linux application in the root namespace opens a virtual accelerator instance and for each new namespace, a group can be created.
   1. Ideally for this case, for fair arbitration, each namespace or group should have exclusive access to a virtual accelerator. (e.g: Application 1 in Figure 3)
   2. Alternatively, one virtual accelerator can be assigned to a namespace and several applications within the namespace can make use of the accelerator in a shared mode. (e.g.: Application 2 and Application 3 in Figure 2.)
   3. Alternatively, if application is by itself able to provide fair arbitration to its various users, then, application can use one virtual accelerator for all its users, by establishing several groups- one for each sub-application. (e.g.: Application 4 in Figure 3)
2. Different Applications can have exclusive virtual accelerator instances assigned to them. (e.g. : Application 5 in Figure 3)

### Modes

Typically applications can access the underlying virtual IPsec accelerator in two modes –namely Exclusive mode or Shared Mode

* Exclusive Mode – The application or sub-application has exclusive access to the Virtual Accelerator Instance. The Virtqueues and any hardware resources allocated to that virtual accelerator are available for the application or sub-application in an exclusive mode.
* Shared Mode – The application or sub-application may share a virtual accelerator with other applications or sub-applications. The Virtqueues and any hardware resources allocated for the virtual accelerator are shared across several applications.

### Virtual Accelerator Assignment

Though now shown in the figure, it is possible that a single application/sub-application may use several virtual accelerators in shared mode or exclusive mode.

The Application or sub-application that has access to several virtual accelerators potentially could distribute the load across the virtual accelerators. The load distribution is entirely up to the application.

# 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\_ipsec\_la\_get\_api\_version](#_g_ipsec_la_get_api_version)
2. [g\_ipsec\_la\_open](#_g_ipsec_la_open)
3. [g\_ipsec\_la\_group\_create](#_g_ipsec_la_group_create)
4. [g\_ipsec\_la\_group\_delete](#_g_ipsec_la_group_delete)
5. [g\_ipsec\_la\_close](#_g_ipsec_la_close)
6. [g\_ipsec\_la\_get\_available\_list](#_g_ipsec_la_get_available_list)
7. [g\_ipsec\_la\_get\_active\_list](#_g_ipsec_la_get_active_list())

## 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\_ipsec\_la\_open() function.

### Control or setup APIs

1. [g\_ipsec\_la\_capabilities\_get](#_g_ipsec_la_get_capabilities)
2. [g\_ipsec\_la\_notification\_hooks\_register](#_g_ipsec_la_notification_hooks_regis)
3. [g\_ipsec\_la\_notifications\_hook\_deregister](#_g_ipsec_la_notifications_hook_dereg)
4. [g\_ipsec\_la\_sa\_add](#_g_ipsec_la_sa_add)
5. [g\_ipsec\_la\_sa\_mod](#_g_ipsec_la_sa_mod)
6. [g\_ipsec\_la\_sa\_del](#_g_ipsec_la_sa_del)
7. [g\_ipsec\_la\_sa\_flush](#_g_ipsec_la_sa_flush)
8. [g\_ipsec\_la\_sa\_get](#_g_ipsec_la_sa_get)

### Data Processing APIs

1. [g\_ipsec\_la\_packet\_encap](#_g_ipsec_la_packet_encap)
2. [g\_ipsec\_la\_packet\_decap](#_g_ipsec_la_packet_decap)
3. [g\_ipsec\_la\_multi\_packet\_encap](#_g_ipsec_la_multi_packet_encap)
4. [g\_ipsec\_la\_multi\_packet\_decap](#_g_ipsec_la_multi_packet_decap)

# g-API definitions

## g\_ipsec\_la\_get\_api\_version

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

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

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

\* Return value: SUCCESS (0) or FAILURE (-ve value)

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

\*/

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

## g\_ipsec\_la\_open

int32\_t g\_ipsec\_la\_open(

enum g\_ipsec\_la\_mode mode, /\* Mode = EXCLUSIVE OR SHARED \*/

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

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

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

\* Input/Output : mode = EXCLUSIVE or SHARES, 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 : SUCCESS(0) or FAILURE (-ve value)

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

\*/

An Application shall use this API to open a virtual accelerator in either a shared mode or exclusive mode. When exclusive mode is requested, every attempt would be made to assign a virtual accelerator exclusively for usage by that application. When shared mode is requested, a shared virtual accelerator may be assigned to the application. In case the suggested mode is unavailable (due to non-available virtual accelerator instances,) a failure would be returned.

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\_ipsec\_la\_group\_create

int32\_t g\_ipsec\_la\_group\_create(

struct g\_ipsec\_la\_handle \*handle;

/\* handle should be valid one \*/

struct g\_ipsec\_la\_group\_create\_inargs \*in,

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

struct g\_ipsec\_la\_group\_create\_outargs \*out,

struct g\_ipsec\_la\_resp\_args resp);

/\* Function Name: g\_ipsec\_la\_group\_create

\* Input :

: g\_ipsec\_la\_handle: handle, char \*group\_identity, a

name that identifies an application group,

\* Output : g\_ipsec\_la\_handle;

\* Return Value : SUCCESS(0) or FAILURE (-ve value)

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

\*/

An Application can use this API to create a group within an accelerator handle. The group would use the same virtual accelerator instance as the one that was assigned as per the application’s g\_ipsec\_la\_open(). Depending on the mode used at the time of g\_ipsec\_la\_open(), the group may be sharing the virtual accelerator instance across several other groups (g\_ipsec\_la\_open() invoked with G\_IPSEC\_LA\_INSTANCE\_EXCLUSIVE), or may be sharing the virtual accelerator across other applications and other groups. (g\_ipsec\_la\_open invoked with G\_IPSEC\_LA\_INSTANCE\_SHARED).

## g\_ipsec\_la\_group\_delete

int32\_t g\_ipsec\_la\_group\_delete(

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

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

struct g\_ipsec\_la\_group\_delete\_outargs \*out,

struct g\_ipsec\_la\_resp\_args resp

)

/\* Function Name: g\_ipsec\_la\_group\_delete

\* Input : accelerator handle and group handle

\* Output : None

\* Return Value : Success(0) or Failure (-ve value)

\* Description : Given a handle, close the group

\*/

Application should use this API to delete a group. Any data structures that were created using this group would be deleted at that point. Application must exercise the g\_ipsec\_la\_sa\_flush API to flush any SAs created with this group, before exercising this call. Application may no longer use the group handle for subsequent calls.

## g\_ipsec\_la\_close

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

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

\* Input : g\_ipsec\_la\_handle; handle

\* Output : None

\* Return Value : Success(0) or Failure (-ve value)

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

Application should use this API to close the handle of the previously opened accelerator instance. If any groups were created under this handle, then the Application should delete them, before making this call. Application must flush all SAs created using the accelerator handle/groups before making this call. Application may no longer access the underlying accelerator.

## g\_ipsec\_la\_avail\_devices\_get\_num

Prototype:

int32 g\_ipsec\_la\_avail\_devices\_get\_num(u32 \*nr\_devices)

/\*

\* Function Name: g\_ipsec\_la\_avai\_devices\_get\_num

\* Input/Output: u32 pointer to hold the value

Return Value: Success or Failure

Description: Return the number of avail virtual accelerator IPsec devices.

\*/

## g\_ipsec\_la\_avail\_devices\_get\_info

Prototype:

int32 g\_ipsec\_la\_avail\_devices\_get\_info(

struct g\_ipsec\_la\_avail\_devices\_get\_inargs \*in,

struct g\_ipsec\_la\_avail\_devices\_get\_outargs \*out)

/\*

\* Function Name: g\_ipsec\_la\_avail\_devices\_get\_info

\* Input: g\_ipsec\_la\_avail\_devices\_get\_in\_args, number of devices for which information has to be retrieved.

\* Ouput: For each available device, get the name, its mode (available or already shared and if shared, the number of apps sharing the device

Description: Application can call this API to find out the list of available devices that it can use.

\*/

## g\_ipsec\_la\_active\_list\_get

* for each virtual accelerator
  + Name
  + the accelerator instance Identifier
  + Application owner details (Application Identity, Group-identity)
  + the mode (shared/exclusive)
  + Displays the statistics
    - Bytes In/Bytes Out
    - Packets In/Packets Out
    - SAs created, SAs Deleted, SAs Modified
    - Current Number of Active SAs

## g\_ipsec\_la\_capabilities\_get

int32\_t **g\_ipsec\_la\_capabilities\_get**(

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

struct g\_ipsec\_la\_control\_flags flags,

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

struct g\_ipsec\_la\_resp\_args \*resp)

/\*

\* Function Name: g\_ipsec\_la\_capabilities\_get

\* Input: handle – accelerator handle with optional group handle;

subflags 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

\*/

Description: Application can call this API to find out the capabilities offered by the underlying virtual IPSec 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\_ipsec\_la\_cap\_get\_outargs through the response callback function.

## g\_ipsec\_la\_notification\_hooks\_register

**int32\_t g\_ipsec\_la\_notification\_hooks\_register**(

struct g\_ipsec\_la\_handle handle, /\* Accelerator Handle \*/

const struct g\_ipsec\_la\_notification\_hooks \*in

);

/\* Function Name: g\_ipsec\_la\_notification\_hooks\_register

\* Input: Virtual Accelerator Instance Handle, Notification hook

functions

\* Output: Success or Failure

\* Description: Registers hook function to be called for notifications

from underlying accelerator; Notifications if supported

by underlying Virtual IPsec accelerator include

Periodic Sequence Number Announce, Sequence Number

Overflow and Soft Lifetime in bytes expiry

\*/

## g\_ipsec\_la\_notifications\_hook\_deregister

g\_ipsec\_la\_notifications\_hook\_deregister(

struct g\_ipsec\_la\_handle , /\* Accelerator Handle \*/ )

/\* Function Name: g\_ipsec\_la\_notifications\_hook\_deregister

\* Input: Accelerator handle, group handle if applicable

\* Ouput: None

\* Description : The notification callback function hooks get de-registered

\*/

Application can call this API to de-register previously registered callback functions.

## g\_ipsec\_la\_sa\_add

int32\_t g\_ipsec\_la\_sa\_add(

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

const struct g\_ipsec\_la\_sa\_add\_inargs \*in,

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

struct g\_ipsec\_la\_sa\_add\_outargs \*out,

struct g\_ipsec\_la\_resp\_args resp);

/\*

\* Function Name: g\_ipsec\_la\_sa\_add

\* Parameters: handle – Accelerator handle,

\* group; Input Arguments = {flags, sa parameters}; flags:

\* Synchronous or asynchronous, Response required or not; Out

\* Argument: Result and SA Handle; resp: Response callback

\* function and callback argument in case ASYNC response is

\* requested

\* Return Value: Success or Failure (< 0)

\* Description: Application uses this API to create an Inbound or

\* Outbound SA

\*/

Application can call this API to create an Inbound or Outbound SA. This API returns SUCCESS when the SA has been successfully created by the Virtual Accelerator. A SA Handle is returned by this API. Application is expected to use the SA Handle in subsequent calls such as g\_ipsec\_la\_sa\_modify, g\_ipsec\_la\_sa\_delete, or one of the Read SA commands

## g\_ipsec\_la\_sa\_mod

int32\_t g\_ipsec\_la\_sa\_mod(

struct g\_ipsec\_la\_hanlde \*handle, /\* Accelerator Handle \*/

const struct g\_ipsec\_la\_sa\_mod\_inargs \*in, /\* Input Arguments \*/

g\_ipsec\_la\_control\_flags flags, /\* Control flags: sync/async, response required or not \*/

struct g\_ipsec\_la\_sa\_mod\_outargs \*out, /\* Output Arguments \*/

struct g\_ipsec\_la\_resp\_args resp /\* Response data structure with callback function information and arguments with ASYNC response is requested);

/\* Function Name: g\_ipseC\_la\_sa\_mod

\* Input/Out: Accelerator Handle, SA Handle, SA Modification parameters, API Control flags, Output arguments, Response callback function and arguments, in case ASYNC mode is chosen

\* Return Value: SUCCESS or FAILURE

\* Description: Application uses this API to modify SA parameters such as Local Gateway IP Address/Port, Remote Gateway IP Address/Port and Sequence number information \*/

Application can call this API to modify SA parameters. When the Local gateway IP Address has been updated or the remote Gateway IP Address has been changed or when sequence number related information has to be updated, Application can call this API to update the SA maintained by the underlying virtual accelerator.

## g\_ipsec\_la\_sa\_del

int32\_t g\_ipsec\_la\_sa\_del(

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

const struct g\_ipsec\_la\_sa\_del\_inargs \*in,

g\_api\_control\_flags flags,

struct g\_ipsec\_la\_sa\_del\_outargs \*out,

struct g\_ipsec\_la\_resp\_args resp);

/\* Function Name: g\_ipsec\_la\_sa\_del

\* Input: Accelerator Handle, SA Direction, SA Handle

\* Input/Output: Success or error code

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

\*/

Application calls this API to delete the SA.

## g\_ipsec\_la\_sa\_flush

Prototype:

int32\_t g\_ipsec\_la\_sa\_flush(

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

g\_ipsec\_la\_control\_flags flags,

struct g\_ipsec\_la\_sa\_flush\_outargs \*out,

struct g\_ipsec\_la\_resp\_args \*resp)

/\* Function Name: g\_ipsec\_la\_sa\_flush

\* Input: Virtual Accelerator Handle and optional group handle

\* information, flags : Async/sync, Response required or not;

\* Response Callback function and argument in case async

\* response is requested

\* Return Value: Success or Failure

\* Description: Application can use this API, to flush the SAs that

\* were created given a handle/group

\*/

Application/sub-application can call this API to flush SAs. If an application has several groups, the application has to flush SAs for each group individually.

## g\_ipsec\_la\_sa\_get

int32\_t g\_ipsec\_la\_sa\_get(

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

const struct g\_ipsec\_la\_sa\_get\_inargs \*in,

g\_ipsec\_la\_\_control\_flags flags,

struct g\_ipsec\_sa\_get\_outargs \*out,

struct g\_ipsec\_la\_\_resp\_args \*resp);

/\* Function Name: g\_ipsec\_la\_sa\_get

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

\* arguments that include direction (inbound or outbound)

\* sa\_handle (valid for get exact or get next calls), Operation

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

\* of SAs to read (for Get First, Get Next and Get Exact, it

\* would be 1; flags: API control flags, out: contains required

\* memory to hold the output information (statistics or SA),

\* 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

\* SA Information or statistics

\*/

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

## g\_ipsec\_la\_packet\_encap

Prototype:

int32\_t g\_ipsec\_la\_packet\_encap(

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

struct g\_ipsec\_la\_control\_flags flags,

struct g\_ipsec\_la\_sa\_handle \*handle, /\* SA Handle \*/

uint32\_t num\_sg\_elem, /\* num of Scatter Gather elements \*/

struct g\_ipsec\_la\_data in\_data[],

/\* Array of data blocks \*/

struct g\_ipsec\_la\_data out\_data[],

/\* Array of output data blocks \*/

struct g\_ipsec\_la\_resp\_args resp)

/\*

\* Function Name: g\_ipsec\_la\_encap\_packet

\* Arguments: Accelerator handle, Control Flags, SA Handle, Input data-

length segments, Output data-length segments, result

\* Success or error code, Response callback and args, in case

\* async response is requested.

\* Return Value : Success or Failure

\*/

Application calls this API for Outbound Packet processing. When the application submits the SA Handle, and the set of input buffers to the virtual accelerator (using handle and optional group), the application expects the virtual accelerator to IPSec outbound process the buffers as per the Security Association and return the processed buffers.

## g\_ipsec\_la\_packet\_decap

Prototype:

int32\_t g\_ipsec\_la\_decap\_packet(

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

struct g\_ipsec\_la\_control\_flags flags,

struct g\_ipsec\_la\_sa\_handle \*handle, /\* SA Handle \*/

uint32\_t num\_sg\_elem, /\* number of Scatter Gather elements \*/

struct g\_ipsec\_la\_data in\_data[],/\* Array of data blocks \*/

struct g\_ipsec\_la\_data out\_data[], /\* Array of out data blocks\*/

struct g\_ipsec\_la\_\_resp\_args resp)

/\*

\* Function Name: g\_ipsec\_la\_decap\_packet

\* Arguments: Accelerator handle, Control Flags, SA,

\* Handle, Input data-length segments

\* Success or error code, array of data blocks to hold the

\* output data, Response callback and args, in case async

\* response is requested.

\* Return Value: Success or Failure

\*/

Application calls this API for Inbound Packet processing. When the application submits the SA Handle, and the set of input buffers to the virtual accelerator (using handle – and optional group), the application expects the virtual accelerator to IPSec inbound process(decapsulation and decryption) the buffers as per the Security Association and return the processed buffers.

## g\_ipsec\_la\_multi\_packet\_encap

Prototype:

int32\_t g\_ipsec\_la\_multi\_packet\_encap(

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

struct g\_ipsec\_la\_control\_flags flags,

struct g\_ipsec\_la\_sa\_handle \*handle; /\* SA Handle \*/

uint32\_t num\_packets; /\* num of Scatter Gather elements \*/

struct g\_ipsec\_la\_packet in\_packets[];

/\* Array of data blocks \*/

struct g\_ipsec\_la\_packet out\_packets[];

/\* Array of output data blocks \*/

struct g\_api\_resp\_args \*resp)

/\*

\* Function Name: g\_ipsec\_la\_encap\_packet

\* Arguments: Accelerator handle, Control Flags, SA Handle, Input

\* packets, Output packets, result

\* Success or error code, Response callback and args, in case

\* async response is requested.

\* Return Value : Success or Failure

\*/

This function is similar to g\_ipsec\_la\_packet\_encap. However multiple packets can be submitted by application in one API invocation.

## g\_ipsec\_la\_multi\_packet\_decap

Prototype:

int32\_t g\_ipsec\_la\_decap\_packet(

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

struct g\_ipsec\_la\_control\_flags flags,

struct g\_ipsec\_la\_sa\_handle \*handle; /\* SA Handle \*/

uint32\_t num\_packets; /\* number of Scatter Gather elements \*/

struct g\_ipsec\_la\_data in\_packets[];/\* Array of in packets \*/

struct g\_ipsec\_la\_data out\_packets[] /\* Array of out packets\*/

struct g\_api\_resp\_args resp)

/\*

\* Function Name: g\_ipsec\_la\_decap\_packet

\* Arguments: Accelerator handle, Control Flags, SA,

\* Handle, Input packets

\* array of packets to hold the

\* processed data, Response callback and args, in case async

\* response is requested.

\* Return Value: Success or Failure

\*/

This function is similar to g\_ipsec\_la\_packet\_decap. However multiple packets can be submitted by application in one API invocation.

# Data Structures

## g\_ipsec\_la\_create\_group\_inargs

struct g\_ipsec\_la\_create\_group\_inargs {

char \*group\_identity; /\* Group identity \*/

}

## g\_ipsec\_la\_create\_group\_outargs

struct g\_ipsec\_la\_create\_group\_outargs {

int32\_t result;

uint32\_t g\_ipsec\_la\_group\_handle[G\_IPSEC\_LA\_GROUP\_HANDLE\_SIZE]; /\* Group handle holder \*/

};

## g\_ipsec\_la\_group\_delete\_outargs

struct g\_ipsec\_la\_group\_delete\_outargs {

int32\_t result;

};

## g\_ipsec\_la\_instance\_broken\_cbk\_fn

typedef void (\*g\_ipsec\_la\_instance\_broken\_cbk\_fn)(struct g\_ipsec\_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.

## g\_ipsec\_la\_open\_inargs

struct g\_ipsec\_la\_open\_inargs {

uint16\_t pci\_vendor\_id; /\* 0x1AF4 \*/

uint16\_t device\_id; /\* Device Id for IPsec \*/

char \*accl\_name; /\* Optional \*/

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

g\_ipsec\_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 \*/

};

## g\_ipsec\_la\_open \_outargs

struct g\_ipsec\_la\_open\_outargs{

g\_ipsec\_la\_handle \*handle /\* handle \*/

};

## g\_ipsec\_la\_resp\_args

struct **g\_ipsec\_la\_resp\_args**

{

struct g\_ipsec\_la\_resp\_cbfn cb\_fn;

/\* Callback function if

ASYNC flag is chosen \*/

void \*cb\_arg;

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

}

The above structure can be used by applications to provide callback function, arguments, that can be subsequently invoked by virtio-ipsec

## g\_ipsec\_la\_handle

struct **g\_ipsec\_la\_handle** {

uint32\_t handle[G\_IPSEC\_LA\_HANDLE\_SIZE]; /\* Accelerator handle \*/

uint32\_t group\_handle[G\_IPSEC\_LA\_GROUP\_HANDLE\_SIZE]; /\* Group handle \*/

};

## g\_ipsec\_la\_avail\_devices\_get\_inargs

struct

{

uint32 num\_devices;

char \*last\_device\_read; /\* NULL if this is the first time this call is invoked; Subsequent calls will have a valid value here \*/

};

## g\_ipsec\_la\_device\_info

struct g\_ipsec\_la\_device\_info

{

char device\_name[IPSEC\_IFNAMESIZ];

u8 mode; /\* Shared or Available \*/

u32 num\_apps; /\* If shared \*/

};

## g\_ipsec\_la\_avail\_devices\_get\_outargs

struct g\_ipsec\_la\_avail\_devices\_get\_outargs

{

uint32 num\_devices; /\* filled by API \*/

/\* Array of pointers, where each points to

device specific information \*/

struct g\_ipsec\_la\_device\_info \*dev\_info;

char \*last\_device\_read; /\* Send a value that the application can use and invoke for the next set of devices \*/

bool b\_more\_devices;

};

## g\_ipsec\_la\_sa\_handle

struct g\_ipsec\_la\_sa\_handle {

uint32\_t ipsec\_sa\_handle[G\_IPSEC\_LA\_SA\_HANDLE\_SIZE];

};

The above structure would be used by the application for all functions once accelerator handle/group handle have been established

## g\_ipsec\_la\_auth\_algo\_cap

/\* Authentication Algorithm capabilities \*/

struct **g\_ipsec\_la\_auth\_algo\_cap** {

uint32\_t md5:1,

sha1:1,

sha2:1,

aes\_xcbc:1,

none:1

des:1;

};

## g\_ipsec\_la\_cipher\_algo\_cap

/\* Cipher Algorithm Capabilities \*/

struct **g\_ipsec\_la\_cipher\_algo\_cap** {

uint32\_t des:1,

des\_c:1,

aes:1,

aes\_ctr:1,

null:1;

};

## g\_ipsec\_la\_comb\_algo\_cap

/\* Combined mode algorithm capabilities \*/

struct **g\_ipsec\_la\_comb\_algo\_cap** {

uint32\_t aes\_ccm:1,

aes\_gcm:1,

aes\_gmac:1;

};

## g\_ipsec\_la\_capabilities

/\* Accelerator capabilities \*/

struct **g\_ipsec\_la\_capabilities**

{

uint32\_t sg\_features:1, /\* Scatter-Gather Support for I/O \*/

ah\_protocol:1, /\* AH Protocol \*/

esp\_protocol:1, /\* ESP protocol \*/

wesp\_protocol:1, /\* WESP Protocol \*/

ipcomp\_protocol:1, /\* IP Compression \*/

multi\_sec\_protocol:1, /\* SA Bundle support \*/

udp\_encap:1, /\* UDP Encapsulation \*/

esn:1, /\* Extended Sequence Number support \*/

tfc:1, /\* Traffic Flow Confidentiality \*/

ecn:1, /\* Extended Congestion Notification \*/

df:1, /\* Fragment bit handling \*/

anti\_replay\_check:1, /\* Anti Replay check \*/

ipv6\_support:1, /\* IPv6 Support \*/

soft\_lifetime\_bytes\_notify:1, /\* Soft Lifetime Notify Support \*/

seqnum\_overflow\_notify:1, /\* Seq Num Overflow notify \*/

seqnum\_periodic\_notify:1; /\* Seq Num Periodic Notify \*/

uint8\_t num\_dscp\_based\_queues; /\* Number of DSCP based queues \*/

struct g\_ipsec\_la\_autho\_algo\_cap auth\_algo\_caps;

struct g\_ipsec\_la\_cipher\_algo\_cap cipher\_algo\_caps;

struct g\_ipsec\_la\_comb\_algo\_cap comb\_algo\_caps;

}

## g\_ipsec\_la\_cap\_get\_outargs

struct **g\_ipsec\_la\_cap\_get\_outargs**

{

struct g\_ipsec\_la\_capabilities caps; /\* Capabilities \*/

}

## g\_ipsec\_la\_resp\_cbfn

typedef void(\***g\_ipsec\_la\_resp\_cbfn**) (void \*cb\_arg, int32\_t cb\_arg\_len, void \*outargs);

## g\_ipsec\_seq\_number\_notification

struct **g\_ipsec\_seq\_number\_notification** {

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

struct g\_ipsec\_la\_sa\_handle \*sa\_handle; /\* SA Handle \*/

uint32\_t seq\_num; /\* Low Sequence Number \*/

uint32\_t hi\_seq\_num; /\* High Sequence Number \*/

};

## g\_ipsec\_la\_cbk\_sa\_seq\_number\_overflow\_fn

/\* Callback function prototype that application can provide to receive sequence number overflow notifications from underlying accelerator \*/

typedef void (\***g\_ipsec\_la\_cbk\_sa\_seq\_number\_overflow\_fn**) (

struct g\_ipsec\_la\_handle handle,

struct g\_ipsec\_seq\_number\_notification \*in);

## g\_ipsec\_la\_cbk\_sa\_seq\_number\_periodic\_update\_fn

/\* Callback function prototype that application can provide to receive sequence number periodic notifications from underlying accelerator \*/

typedef void (\***g\_ipsec\_la\_cbk\_sa\_seq\_number\_periodic\_update\_fn**) (

struct g\_ipsec\_la\_handle handle,

struct g\_ipsec\_seq\_number\_notification \*in);

## g\_ipsec\_la\_lifetime\_in\_bytes\_notification

struct **g\_ipsec\_la\_lifetime\_in\_bytes\_notification** {

struct g\_ipsec\_la\_sa\_handle sa\_handle; /\* SA Handle \*/

uint32\_t ipsec\_lifetime\_in\_kbytes; /\* Lifetime in Kilobytes \*/

}

## g\_ipsec\_la\_cbk\_sa\_soft\_lifetimeout\_expiry\_fn

/\* Callback function prototype that application can provide to receive soft lifetime out expiry from underlying accelerator \*/

typedef void (\***g\_ipsec\_la\_cbk\_sa\_soft\_lifetimeout\_expiry\_fn**) (

struct g\_ipsec\_la\_handle handle,

struct g\_ipsec\_la\_lifetime\_in\_bytes\_notification \*in);

## g\_ipsec\_la\_notification\_hooks

struct **g\_ipsec\_la\_notification\_hooks**

{

/\* Sequence Number Overflow callback function \*/

struct g\_ipsec\_la\_cbk\_sa\_seq\_number\_overflow\_fn \*seq\_num\_overflow\_fn;

/\* Sequence Number periodic Update Callback function \*/

struct g\_ipsec\_la\_cbk\_sa\_seq\_number\_periodic\_update\_fn \*seq\_num\_periodic\_update\_fn;

/\* Soft lifetime in Kilobytes expiry function \*/

struct g\_ipsec\_la\_cbk\_sa\_soft\_lifetimeout\_expiry\_fn \*soft\_lifetimeout\_expirty\_fn;

void \*seqnum\_overflow\_cbarg;

u32 seq\_num\_overflow\_cbarg\_len;

void \*seqnum\_periodic\_cbarg;

u32 seq\_num\_periodic\_cbarg\_len;

void \*soft\_lifetimeout\_cbarg;

u32 soft\_lifetimeout\_cbarg\_len;

};

## g\_ipsec\_la\_sa\_crypto\_params

struct g\_ipsec\_la\_sa\_crypto\_params

{

enum g\_ipsec\_la\_auth\_alg auth\_algo;

uint8\_t \*auth\_key; /\* Authentication Key \*/

uint32\_t auth\_key\_len\_bits; /\* Key Length in bits \*/

enum g\_ipsec\_la\_cipher\_alg cipher\_algo; /\* Cipher Algorithm \*/

uint8\_t \*cipher\_key; /\* Cipher Key \*/

uint32\_t cipher\_key\_len\_bits; /\* Cipher Key Length in bits \*/

uint8\_t icv\_len\_bits; /\* ICV – Integrity check value size in bits \*/

}

## g\_ipsec\_la\_ipcomp\_info

struct g\_ipsec\_la\_ipcomp\_info

{

enum g\_ipsec\_la\_ipcomp\_alg algo;

uint32\_t cpi;

}

## g\_ipsec\_la\_tunnel\_end\_addr

struct g\_ipsec\_la\_tunnel\_end\_addr {

struct g\_ip\_addr src\_ip; /\* Source Address \*/

struct g\_ip\_addr dest\_ip; /\* Destination Address \*/

};

## g\_ipsec\_la\_nat\_traversal\_info

struct g\_ipsec\_la\_nat\_traversal\_info {

uint16\_t dest\_port; /\* Destination Port \*/

uint16\_t src\_port; /\* Source Port \*/

struct g\_ip\_addr nat\_oa\_peer\_addr; /\* Original Peer Address; valid if encapsulation Mode is transport \*/

};

## g\_ipsec\_la\_sa

struct g\_ipsec\_la\_sa

{

uint32\_t spi; /\* Security Parameter Index \*/

uint8\_t proto; /\* ESP, AH or IPCOMP \*/

enum g\_ipsec\_la\_sa\_flags cmn\_flags; /\* Flags such as Anti-replay check, ECN etc \*/

union {

struct {

uint8\_t dscp; /\* DSCP value valid when dscp\_handle is set to “set” \*/

enum g\_ipsec\_la\_df\_handle df\_bit\_handle; /\* DF set, clear or propogate \*/

enum g\_ipsec\_la\_dscp\_handle dscp\_handle; /\* DSCP handle set, clear etc. \*/

}outb;

struct {

//enumg\_ipsec\_la\_inb\_sa\_flags flags; /\* Flags specific to inbound SA \*/

}inb;

}

struct g\_ipsec\_la\_sa\_crypto\_params crypto\_params; /\* Crypto Parameters \*/

struct g\_ipsec\_la\_ipcomp\_info; /\* IP Compression Information \*/

uint32\_t soft\_kilobytes\_limit;

uint32\_t hard\_kilobytes\_limit;

uint32\_t seqnum\_interval;

struct g\_\_ipsec\_la\_nat\_traversal\_info nat\_info;

struct g\_\_ipsec\_la\_tunnel\_end\_addr te\_addr;

}

## g\_ipsec\_la\_sa\_add\_inargs

struct g\_ipsec\_la\_sa\_add\_inargs

{

enum g\_ipsec\_la\_sa\_direction dir;

uint8\_t num\_sas;

struct g\_ipsec\_la\_sa \*sa\_params;

};

## g\_ipsec\_la\_sa\_add\_outargs

struct g\_ipsec\_la\_sa\_add\_outargs {

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

struct g\_ipsec\_la\_handle handle;

}

## g\_ipsec\_la\_sa\_modify\_flags

struct g\_ipsec\_la\_sa\_modify\_flags

{

G\_IPSEC\_LA\_SA\_MODIFY\_LOCAL\_GW\_INFO= 1, /\* Modify the Local Gateway Information \*/

G\_IPSEC\_LA\_SA\_MODIFY\_PEER\_GW\_INFO, /\* Modify the Remote Gateway Information \*/

G\_IPSEC\_LA\_SA\_MODIFY\_REPLAY\_INFO, /\* SA will be updated with Sequence number, window bit map etc. \*/

};

## g\_ipsec\_la\_sa\_mod\_inargs

struct g\_ipsec\_la\_sa\_mod\_inargs

{

enum g\_ipsec\_la\_sa\_direction; /\* Inbound or Outbound \*/

struct g\_ipsec\_la\_sa\_handle \*handle; /\* SA Handle \*/

enum g\_ipsec\_la\_sa\_modify\_flags flags; /\* Flags that indicate what needs to be updated \*/

union {

struct {

uint16\_t port; /\* New Port \*/

struct g\_ip\_addr addr; /\* New IP Address \*/

}addr\_info; /\* Valid when Local or Remote Gateway Information is modified \*/

struct {

enum g\_ipsec\_la\_sa\_modify\_replay\_info\_flags flags; /\* Flag indicates which parameters are being modified \*/

uint8\_t anti\_replay\_window\_size; /\* Anti replay window size is being modified \*/

uint32\_t anti\_replay\_window\_bit\_map; /\* Window bit map array is being updated \*/

uint32\_t seq\_num; /\* Sequence Number is being updated \*/

uint32\_t hi\_seq\_num; /\* Higher order Sequence number, when Extended Sequence number is used \*/

}; /\* Valid when SA\_MODIFY\_REPLAY\_INFO is set \*/

}

};

## g\_ipsec\_la\_sa\_mod\_outargs

struct g\_ipsec\_la\_sa\_mod\_outargs

{

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

}

## g\_ipsec\_la\_sa\_del\_inargs

struct g\_ipsec\_la\_sa\_del\_inargs

{

enum g\_ipsec\_la\_sa\_direction dir; /\* Input or Output \*/

struct g\_ipsec\_la\_sa\_handle \*handle; /\* SA Handle \*/

};

## g\_ipsec\_la\_sa\_del\_outargs

struct g\_ipsec\_la\_sa\_del\_outargs

{

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

};

## g\_ipsec\_la\_sa\_flush\_outargs

struct g\_ipsec\_la\_sa\_flush\_outargs {

int32\_t result; /\* 0 for success \*/

}

## g\_ipsec\_la\_sa\_stats

struct g\_ipsec\_la\_sa\_stats {

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

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

struct {

uint32\_t invalid\_ipsec\_pkt; /\* Number of invalid IPSec Packets \*/

uint32\_t invalid\_pad\_length; /\* Number of packets with invalid padding length \*/

uint32\_t invalid\_seq\_num; /\* Number of packets with invalid sequence number \*/

uint32\_t anti\_replay\_late\_pkt; /\* Number of packets that failed anti-replay check through late arrival \*/

uint32\_t anti\_replay\_replay\_pkt; /\* Number of replayed packets \*/

uint32\_t invalid\_icv; /\* Number of packets with invalid ICV \*/

uint32\_t seq\_num\_over\_flow; /\* Number of packets with sequence number overflow \*/

uint32\_t crypto\_op\_failed; /\* Number of packets where crypto operation failed \*/

}protocol\_violation\_errors;

struct {

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 \*/

}process\_errors;

}

## g\_ipsec\_la\_sa\_get\_outargs

struct g\_ipsec\_la\_sa\_get\_outargs {

{

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

struct g\_ipsec\_la\_sa \*sa\_params; /\* An array of sa\_params[] to hold ‘num\_sas’ information \*/

struct g\_ipsec\_la\_sa\_stats \*stats; /\* An array of stats[] to hold the statistics \*/

g\_ipsec\_la\_sa\_handle \*\* handle; /\* handle returned to be used for subsequent Get Next N call \*/

};

## g\_ipsec\_la\_sa\_get\_inargs

struct g\_ipsec\_la\_sa\_get\_inargs

{

enum g\_ipsec\_la\_sa\_direction dir; /\* Direction: Inbound or Outbound \*/

/\* Following field is not applicable for get\_first \*/

struct g\_ipsec\_la\_sa\_handle \*handle;

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

uint32\_t num\_sas; /\* Number of SAs to read \*/

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

}

## g\_ipsec\_la\_data

struct g\_ipsec\_la\_data {

uint8\_t \*buffer; /\* Buffer pointer \*/

uint32\_t length; /\* Buffer length \*/

}

## g\_ipsec\_la\_packet

struct g\_ipsec\_la\_packet{

uint32\_t num\_sg; /\* Number of scatter gather elements \*/

struct \*g\_ipsec\_la\_data; /\* array of buffer segments \*/

}

## g\_ipsec\_la\_ipv6\_addr

struct g\_ipsec\_la\_ipv6\_addr{

/\*! U8 Addr Len. \*/

#define G\_IPSEC\_LA\_IPV6\_ADDRU8\_LEN 16

/\*! U32 Addr len \*/

#define G\_IPSEC\_LA\_IPV6\_ADDRU32\_LEN 4

union {

uint8\_t b\_addr[G\_IPSEC\_LA\_IPV6\_ADDRU8\_LEN];

/\*\*< Byte addressable v6 address \*/

uint32\_t w\_addr[G\_IPSEC\_LA\_IPV6\_ADDRU32\_LEN];

/\*\*< Word addressable v6 address \*/

};

};

## g\_ipsec\_la\_ip\_addr

struct g\_ipsec\_la\_ip\_addr {

enum g\_ipsec\_la\_ip\_version version; /\*\*< IP Version \*/

/\*! Union details

\*/

union {

uint32\_t ipv4;

/\*\*< IPv4 Address \*/

struct g\_ipsec\_la\_ipv6\_addr ipv6;

/\*\*< IPv6 Address \*/

};

};

# Macros

#define G\_IPSEC\_LA\_HANDLE\_SIZE 8

#define G\_IPSEC\_LA\_GROUP\_HANDLE\_SIZE 8

#define G\_IPSEC\_LA\_SA\_HANDLE\_SIZE 8

# Enumerations

## g\_ipsec\_la\_mode

enum g\_ipsec\_la\_mode {

G\_IPSEC\_LA\_INSTANCE\_AVAILABLE=0, /\* Not assigned to any mode \*/

G\_IPSEC\_LA\_INSTANCE\_EXCLUSIVE=1, /\* Exclusive Mode \*/

G\_IPSEC\_LA\_INSTANCE\_SHARED /\* Shared Mode \*/

};

## g\_ipsec\_la\_control\_flags

enum **g\_ipsec\_la\_control\_flags**

{

G\_IPSEC\_LA\_CTRL\_FLAG\_ASYNC, /\* If Set, API call be asynchronous. Otherwise, API call will be synchronous \*/

G\_IPSEC\_LA\_CTRL\_FLAG\_NO\_RESP\_EXPECTED, /\* 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.

## g\_ipsec\_la\_auth\_alg

enum g\_ipsec\_la\_auth\_alg {

G\_IPSEC\_LA\_AUTH\_ALGO\_NONE=1, /\* No Authentication \*/

G\_IPSEC\_LA\_AUTH\_ALGO\_MD5\_HMAC /\* MD5 HMAC Authentication Algo. \*/

G\_IPSEC\_LA\_AUTH\_ALGO\_SHA1\_HMAC /\* SHA1 HMAC Authentication Algo. \*

G\_IPSEC\_LA\_AUTH\_AESXCBC, /\* AES-XCBC Authentication Algo. \*/

G\_IPSEC\_LA\_AUTH\_ALG\_SHA2\_256\_HMAC /\* SHA2 HMAC Authentication Algorithm; 256 bit key length \*/

G\_IPSEC\_LA\_AUTH\_ALGO\_SHA2\_384\_HMAC /\* SHA2 HMAC Authentication Algorithm with 384 bit key length \*/

G\_IPSEC\_LA\_AUTH\_ALGO\_SHA2\_512\_HMAC /\* SHA2 HMAC Authentication Algorithm with 512 bit key length \*/,

G\_IPSEC\_LA\_AUTH\_ALGO\_HMAC\_SHA1\_160

};

## g\_ipsec\_la\_cipher\_alg

enum g\_ipsec\_la\_cipher\_alg {

G\_IPSEC\_LA\_CIPHER\_ALGO\_NULL=1, /\* NULL Encryption algorithm \*/

G\_IPSEC\_LA\_ALGO\_DES\_CBC, /\* DES-CBC Encryption Algorithm \*/

G\_IPSEC\_LA\_ALGO\_3DES\_CBC,

G\_IPSEC\_LA\_ALGO\_AES\_CBC,

G\_IPSEC\_LA\_ALGO\_AES\_CTR,

G\_IPSEC\_LA\_ALGO\_COMB\_AES\_CCM, /\* AES-CCM \*/

G\_IPSEC\_LA\_ALGO\_COMB\_AES\_GCM, /\* AES-GCM \*/

G\_IPSEC\_LA\_ALGO\_COMB\_AES\_GMAC /\* AES-GMAC \*/

};

## g\_ipsec\_la\_ipcomp\_alg

enum g\_ipsec\_la\_ipcomp\_alg {

G\_IPSEC\_LA\_IPCOMP\_DEFLATE=1, /\* Deflate IP Compression Algorithm \*/

G\_IPSEC\_LA\_IPCOMP\_LZS /\* LZS IP Compression Algorithm \*/

};

## g\_ipsec\_la\_dscp\_handle

enum g\_ipsec\_la\_dscp\_handle {

G\_IPSEC\_LA\_DSCP\_COPY=1, /\* copy from inner header to tunnel outer header \*/

G\_IPSEC\_LA\_DSCP\_CLEAR, /\* Clear the DSCP value in outer header \*/

G\_IPSEC\_LA\_DSCP\_SET, /\* Set the DSCP value in outer header to specific value \*/

};

## g\_ipsec\_la\_df\_handle

enum g\_ipsec\_la\_df\_handle {

G\_IPSEC\_LA\_DF\_COPY=1, /\* Copy DF bit from inner to outer \*/

G\_IPSEC\_LA\_DF\_CLEAR, /\* Clear the DF bit in outer header \*/

G\_IPSEC\_LA\_DF\_SET /\* Set the bit in the outer header \*/

};

## g\_ipsec\_la\_sa\_direction

enum g\_ipsec\_la\_sa\_direction {

G\_IPSEC\_LA\_SA\_INBOUND,

G\_IPSEC\_LA\_SA\_OUTBOUND

};

## g\_ipsec\_la\_sa\_flags

enum g\_ipsec\_la\_sa\_flags

{

G\_IPSEC\_LA\_SA\_DO\_UDP\_ENCAP\_FOR\_NAT\_TRAVERSAL = BIT(1),

G\_IPSEC\_LA\_SA\_USE\_ECN = BIT(2),

G\_IPSEC\_LA\_SA\_LIFETIME\_IN\_KB = BIT(3),

G\_IPSEC\_LA\_SA\_DO\_ANTI\_REPLAY\_CHECK = BIT(4),

G\_IPSEC\_LA\_SA\_ENCAP\_TRANSPORT\_MODE = BIT(5),

G\_IPSEC\_LA\_SA\_USE\_ESN=BIT(6),

G\_IPSEC\_LA\_SA\_USE\_IPv6=BIT(7),

G\_IPSEC\_LA\_NOTIFY\_LIFETIME\_KB\_EXPIRY=BIT(8),

G\_IPSEC\_LA\_NOTIFY\_SEQNUM\_OVERFLOW=BIT(9),

G\_IPSEC\_LA\_NOTIFY\_SEQNUM\_PERIODIC=BIT(10)

};

## g\_ipsec\_la\_inb\_sa\_flags

enum g\_ipsec\_la\_inb\_sa\_flags {

NF\_IPSEC\_INB\_SA\_PROPOGATE\_ECN =1

/\* When set, ENC from outer tunnel packet will be propagated to the decrypted packet \*/

};

## g\_ipsec\_la\_sa\_modify\_replay\_info\_flags

enum g\_ipsec\_la\_sa\_modify\_replay\_info\_flags {

G\_IPSEC\_LA\_SA\_MODIFY\_SEQ\_NUM= BIT(1), /\* Sequence number is being updated \*/

G\_IPSEC\_LA\_SA\_MODIFY\_ANTI\_REPLAY\_WINDOW = BIT(2) /\* Anti-replay window is being updated \*/

};

## g\_ipsec\_la\_sa\_get\_op

enum g\_ipsec\_la\_sa\_get\_op {

G\_IPSEC\_LA\_SA\_GET\_FIRST\_N = 0,

G\_IPSEC\_LA\_SET\_GET\_NEXT\_N,

G\_IPSEC\_LA\_SA\_GET\_EXACT

};

## g\_ipsec\_la\_ip\_version

enum g\_ipsec\_la\_ip\_version {

G\_IPSEC\_LA\_IPV4 = 4, /\*\*< IPv4 Version \*/

G\_IPSEC\_LA\_IPV6 = 6 /\*\*< IPv6 Version \*/

};