# API Guidelines

## Basic Types

Linux type definitions shall be followed. For example int, u64, char etc. shall be used.

## API Naming convention

APIs are named such that they have the accelerator name, type and function. For instance, an ipsec look aside accelerator’s SA creation function shall be named as g\_ipsec\_la\_sa\_add(). ‘ipsec’ refers to the accelerator name, ‘la’ indicates the type as look aside and sa\_add is the actual function. At all times the object precedes the operation, as in this case ‘sa’ precedes ‘add’.

## Variable Naming convention

Naming convention for variables shall follow Linux style, readable and separated by underscore, when necessary.

## Function Arguments and Return Values

All APIs return a value of SUCCESS or FAILURE.

For control or setup APIs that are used to setup states in the hardware accelerator it is preferable to use data structures to pass input and output parameters. While these setup or control functions do not come in the data path and hence do not impact performance, having parameters defined as structures enables extensibility in future without changing API prototypes. Structure introduced for passing in as parameters for functions shall have the function name as prefix and inargs/outargs as suffixes to indicate input and output arguments. For example, the input argument to g\_ipsec\_la\_sa\_add() would be g\_ipsec\_la\_sa\_add\_inargs and g\_ipsec\_la\_sa\_add\_outargs.

For data processing APIs, data structures are avoided in the packet processing calls and linear buffers are used with performance considerations in mind.

APIs shall also have flags to modify API behavior such as synchronous/asynchronous, response expected or not.

For example a set up API for setting up SAs would be as follows:

int g\_ipsec\_la\_sa\_add(

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

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

enum g\_ipsec\_la\_control\_flags flags, /\* API flags \*/

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

struct g\_ipsec\_la\_resp\_args resp /\* response callback in case

asynchronous mode with response flag is set \*/ );

In the above API, g\_ipsec\_la\_control\_flags and g\_ipsec\_la\_resp\_args are defined as follows:

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 \*/

}**;**

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 \*/

}

Application can request the response to be returned synchronously or asynchronously (G\_IPSEC\_LA\_CTRL\_FLAG\_ASYNC). If the response is requested asynchronously, then the application should provide a callback function pointer and callback argument.

Also, in some scenarios, the API layer may have to do additional operations to force a response from the backend. The flag G\_IPSEC\_LA\_CTRL\_FLAG\_NO\_RESP\_EXPECTED can be used by application to indicate whether the application should force the response from the backend or not.

A packet processing API in the case of IPSec would be as follows:

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 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\_api\_resp\_args resp)

In the above API, g\_ipsec\_la\_data is defined as follows:

struct g\_ipsec\_la\_data {

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

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

}

## API Types

APIs can be classified as management APIs and functional APIs.

Management APIs include APIs that VNF applications can use to find out about available accelerators, accelerator usage request and relinquish.

Functional APIs include Control or setup APIs for setting up state in the stateful hardware accelerator and and data processing APIs for packet processing.

Typically for any stateful hardware accelerator, the following APIs would be made available for control or setup of states

*add* – To add a state in the hardware accelerator

*mod* – To modify a state in the hardware accelerator

*del* – To delete a state in the hardware accelerator

*get* – get the current state as seen and maintained by the hardware accelerator; get types include get-first, get-next, get-exact etc.

## Example: g-APIs for IPsec

G-APIs for IPsec are defined to allow VNF application access underlying hardware accelerator to perform IPSec accelerator operations.

G-APIs for IPSec shall include the following:

g\_ipsec\_la\_open(), g\_ipsec\_la\_close(), g\_ipsec\_la\_sa\_add(), g\_ipsec\_la\_sa\_del(), g\_ipsec\_la\_sa\_mod(), g\_ipsec\_la\_sa\_get(), g\_ipsec\_la\_packet\_encap(), g\_ipsec\_la\_packet\_decap(), g\_ipsec\_la\_multi\_packet\_encap(), g\_ipsec\_la\_multi\_packet\_decap().