IP security (IPsec) Service API Implementation Agreement

August 2004
Revision 1.0

Editors:

Ken Grewal, Intel Corporation, ken.grewal@intel.com

Suman Sharma, Intel Corporation, suman.sharma@intel.com

The words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", “MAY", and "OPTIONAL" in the remainder of this document are to be interpreted as described in the NPF Software API Conventions Implementation Agreement revision 1.0.

For additional information contact:
The Network Processing Forum, 39355 California Street,
Suite 307, Fremont, CA 94538
+1 510 608-5990 phone ✦ info@npforum.org

TABLE OF CONTENTS

1 REVISION HISTORY 3

2 NORMATIVE REFERENCES 4

3 ACRONYMS AND ABBEREVIATIONS 5

4 INTRODUCTION 6

4.1 ASSUMPTIONS & EXTERNAL REQUIREMENTS 7
4.2 SCOPE 7
4.3 DEPENDENCIES 8

5 DATA TYPES 9

5.1 IPSEC SERVICE API TYPES 9
5.2 DATA STRUCTURES FOR COMPLETION CALLBACKS 17
5.3 DATA STRUCTURES FOR EVENT NOTIFICATIONS 22
5.4 ERROR CODES 25
5.5 CORE DEFINITIONS 28

6 FUNCTIONS 30

6.1 COMPLETION CALLBACK 30
6.2 EVENT NOTIFICATION 32
6.3 EVENT DEFINITION SIGNATURE 34
6.4 COMPLETION CALLBACKS AND ERROR RETURNS 34
6.5 ORDER OF OPERATIONS 35
6.6 IPSEC SERVICE API 35

7 API CAPABILITIES 71

7.1 API FUNCTIONS 71
7.2 API EVENTS 71

APPENDIX A INFORMATIVE ANNEXES 73

A.1. HEADER FILE 74

APPENDIX B ACKNOWLEDGEMENTS 100

APPENDIX C LIST OF COMPANIES BELONGING TO NPF DURING APPROVAL PROCESS 101

Table of Figures

FIGURE 1: IPSEC - NPF APIS ARCHITECTURAL RELATIONSHIP. 6

1 Revision History

Revision	Date	Reason for Changes
1.0	08/05/2004	Created Rev 1.0 of the implementation agreement by taking the IP Security Service API (npf2003.259.23) and making minor editorial corrections.

2 Normative References

NPF API (Acronym)	Revision	Description
NPF SAPI Conv.	2.0	Software API Conventions, Implementation Agreement
NPF Lexicon	1.0	NPF API Framework Lexicon, Implementation Agreement
NPF Framework	1.0	NPF API Framework, Implementation Agreement
NPF IM API	2.0	NPF Interface Management API, Implementation Agreement
NPF IPv4 UFwd	2.0	NPF IPv4 Unicast Forwarding Service API, Implementation Agreement
NPF IPv6 UFwd	2.0	NPF IPv6 Unicast Forwarding Service API, Implementation Agreement
IETF RFC	**Revision**	**Description**
2104	Final	H. Krawczyk, M. Bellare and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. February 1997. IETF.
2367	Final	D. McDonald, C. Metz, B. Phan. PF_KEY Key Management API, Version 2, July 1998. IETF.
2393	Final	A. Shacham et. al. IP Payload Compression Protocol (IPComp). December 1998. IETF.
2401	Final	S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. November 1998. IETF.
2402	Final	S. Kent and R. Atkinson. IP Authentication Header. November 1998. IETF.
2403	Final	C. Madson and R. Glenn. The Use of HMAC-MD5-96 within ESP and AH. November 1998. IETF.
2404	Final	C. Madson and R. Glenn. The Use of HMAC-SHA-1-96 within ESP and AH. November 1998.IETF
2405	Final	C. Madson and N. Doraswamy. The ESP DES-CBC Cipher Algorithm With Explicit IV. November 1998. IETF.
2406	Final	S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). November 1998. IETF.
2407	Final	D. Piper. The Internet IP Security Domain of Interpretation for ISAKMP. November 1998. IETF.
2408	Final	D. Maughan et. al. Internet Security Association and Key Management Protocol (ISAKMP). November 1998. IETF.
2409	Final	D. Harkins and D. Carrel. The Internet Key Exchange (IKE). November 1998. IETF.
2410	Final	R. Glenn and S. Kent. The NULL Encryption Algorithm and Its Use With IPSec. November 1998. IETF
2411	Final	R. Thayer, N. Doraswamy and R. Glenn. IP Security Document Readmap. November 1998. IETF.
2412	Final	H. Orman. The OAKLEY Key Determination Protocol. November 1998. IETF.
2451	Final	R. Pereira. The ESP CBC-Mode Cipher Algorithms. November 1998. IETF.

3 Acronyms and Abbereviations

The following acrynoms and abbereviations are used in the specifications:

API	Applications Programming Interface
FAPI	NPF Functional API
FIB	Forwarding Information Base
GW	Gateway
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IP	Internet Protocol
IPSec	IP Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
KMD	Key Management Daemon
LFB	Logical Functional Block
NE	Network Element
NP	Network Processor
NPE	Network Processing Element
NPF	Network Processing Forum
NPU	Network Processing Unit
RFC	Request For Comments (IETF standard)
RIB	Routing Information Base
SA	Security Association
SAD	Security Association Database
SPI	Security Parameter Index (used for ESP / AH SAs)
CPI	Compression Parameter Index (used for IPCOMP SAs)
SAPI	NPF Service API
SCTP	Stream Control Transmission Protocol
SPD	Security Policy Database
UI	User Interface
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
IM API	NPF Interface Management API
PH API	NPF Packet Handler API

4 Introduction

IPSec provides security services at the IP layer. These services include data confidentiality, integrity and authentication. In order to provide these services, two rules databases need to be defined in accordance with the IPSec protocol specification. These are the Security Policy Database (SPD) and the Security Association Database (SAD). These databases contain various attributes allowing a given IPSec implementation in the forwarding plane to determine how to handle ingress and egress IP data packets. Within the IPSec realm, the SPD defines what to do in handling a given IP packet, whereas the SAD defines how to do this.

The IPSec Service API (SAPI) provides a generic interface for configuring and managing the IPSec databases (SPD and SAD). Furthermore, the IPSec SAPI allows a client application to receive event notifications indicating state changes, alerts and other information data. In accordance with the NPF framework model, the IPSec SAPI is Network Element unaware.

The following diagram depicts the typical architecture / relationship between the IPSec SAPI and higher / lower layer components, in accordance with the NPF API framework.

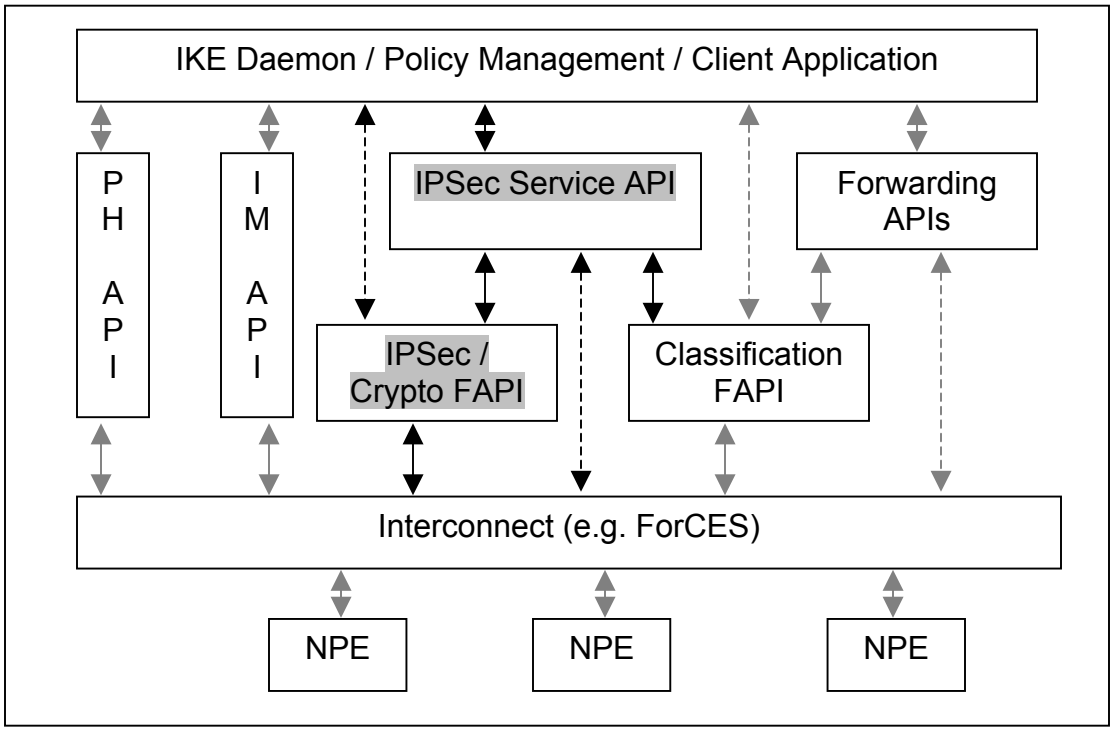


Figure 1: IPSec - NPF APIs architectural relationship.

The IPSec specific APIs are depicted as grey in the above diagram.

In order to utilize IPSec via the IPSec Service API in the NPF framework, the client application may need to interface with other NPF Operational and Service APIs. These are primarily the Interface Management API, the Packet Handler API and the IPv4 / v6 Forwarding APIs. The Interface Management API is always used to setup interfaces to which SPDs can be bound. The Packet Handler API is exploited by IKE for receiving and sending ISAKMP messages to the data path and ultimately to the peer.

A possible implementation of the IPSec Service API is also depicted in the figure. Such an implementation could use the Generic Classification FAPI for configuring selector filters in individual NPEs while using the IPSec/Crypto FAPI for passing the NPEs relevant keys, IPSec header and mode (tunnel/transport), crypto algorithms etc.

The figure additionally shows how an application can bypass the IPSec SAPI, should there be a need for it. E.g. this could happen if the application detects a memory shortage somewhere in the system and wants to free memory at a specific NPE. Additionally, the IPSec FAPI / Crypto FAPI may be used directly to interact with individual FEs and install the required IPSec rules.

The sequence of operations required to configure IPSec will depend on the application. Below are a typical set of steps needed to configure IPSec.

- Create an interface and set the corresponding interface parameters using the IM API.
- Create the IPSec SPD databases (SPD) and bind to the interface.
- Add the IPSec policy / SA information into the IPSec databases (SPD and SAD).
- Setup appropriate rules in the internal / external interface FIB, allowing traffic to be routed correctly.

4.1 Assumptions & External Requirements

- In the scope of this API, the structure and attributes of IPSec reflect what is needed by the forwarding plane, not by the application. Applications are expected to maintain their own representation of the IPSec databases, very likely in a system that permits more and different attributes than recognized by the API. These additional attributes will likely pertain to the data required by the Key Management Daemon (KMD), i.e. IKE, in the negotiation of the IPSec SAs, as well as a more detailed policy database.
- Memory allocation and usage model for this API implementation will be as dictated by the NPF Software Conventions Implementation Agreement.
- The IPSec SAPI is designed in accordance with the NPF framework design principle. Usually control applications need assistance from a number of APIs. In particular the Operational APIs – such as the Interface Management API and the Packet Handler API – are needed by most applications. IPSec is no exception. For a usable system, the IPSec SAPI is also reliant on the interface management API. Additionally, if a key management daemon was running on the system, it would require the services of the packet handler API.
- Binding of data structures to interfaces is done through the IPSec SAPI. This is different from the case with the IPv4/IPv6 SAPIs where a function in the interface management API is used to bind a FIB.

4.2 Scope

The IPSec SAPI supports the IPSec attributes as required by the forwarding plane and does not cover any other attributes that may be required in either packet processing or the negotiation of the IPSec SAs. These additional attributes are considered outside the scope of this document.

The IPSec SAPI will conform to the standards / usage as described in the NPF foundations document and hence will provide the NPF method of asynchronous callback completion and event handling.

The IPSec SAPI contains no operations that deal explicitly with configuration of tunnels where one is nested in the other and both either originate or terminate in the same NE. General nesting may involve one-to-one, many-to-one or many-to-many relationships between the SAs used for inner and outer protections, but the SAPI lacks the means of explicitly expressing such relationships. Note, though, that an implementation may still support implicit nesting by allowing packets to iterate through IPSec processing – corresponding to an, in principle, all-to-all relation between inner and outer SAs. Also note that full support is retained for combining SAs using transport adjacency.

Apart from the lack of support for nested tunnels, the IPSec SAPI supports all aspects of [RFC 2401]. Additionally, the SAPI allows configuration of multiple VPNs within an NE – a feature that is not directly treated in [RFC 2401] and which requires a slight extension to the RFC. Similarly the IPSec SAPI supports multiple selectors per policy/SA – a feature needed to secure SCTP sessions while using a minimal number of SAs.

4.3 Dependencies

The IPSec SAPI utilizes conventions from the following NPF documents.

1. [NPF Lexicon]
2. [NPF SAPI conv]
3. [NPF Framework]
4. [NPF IM API]
5. [NPF IPv4 UFwd]
6. [NPF IPv6 UFwd]
7. [NPF Global]

Furthermore, the IPSec SAPI is closely coupled with [NPF IM API], [NPF IPv4 UFwd] and [NPF IPv6 UFwd]. These APIs are required to create an interface (physical or logical), associated FIBs, as well as binding these to an interface. The resultant interface handle is explicitly used to bind the IPSec security data and rules to a given interface via the IPSec SAPI.

IPv4 and IPv6 addresses come from [NPF Global].

The IPSec SAPI is based on the IPSec protocol suite described in the IPSec RFCs 2401 - 2412. IP Payload Compression is based on [RFC 2393]. Concepts from those RFCs are used repeatedly throughout this document.

5 Data types

5.1 IPSec Service API Types

5.1.1 Interface Handle Array : NPF_IPSecInterface_Handle_Array_t

```
/*
*     Interface Handle Array
*/
typedef struct
{
      NPF_uint32_t            nifHandles;
      NPF_IfHandle_t          *ifHandleArray;

} NPF_IPSecInterface_Handle_Array_t;
```

5.1.2 IPSec Direction : NPF_IPSecDirection_t

```
/*
*     The Direction of a given policy / SA
*/
typedef enum
{
      NPF_IPSEC_DIRECTION_INBOUND=1,      /* Inbound policy / SA */
      NPF_IPSEC_DIRECTION_OUTBOUND=2      /* Outbound policy / SA */
} NPF_IPSecDirection_t;
```

5.1.3 SPD ID : NPF_IPSecSPD_ID_t

```
/*
*     SPD ID
*
*     SPD identity selected by the caller
*/
typedef NPF_uint32_t NPF_IPSecSPD_ID_t; /* SPD ID */
```

5.1.4 SPD Handle : NPF_IPSecSPD_Handle_t

```
/*
*     SPD Handle
*
*     SPD handle selected by the implementation
*/
typedef NPF_uint32_t NPF_IPSecSPD_Handle_t; /* SPD Handle */

/* The following values may not be used to indicate a valid SPD handle */
#define NPF_IPSEC_SPD_HANDLE_INVALID 0
```

5.1.5 SPD Handle Array : NPF_IPSecSPD_Handle_Array_t

```
/*
*     IPSec SPD Handle Array
*
*     SPD handles selected by the implementation
*/
typedef struct
{
      NPF_uint32_t nCount;
      NPF_IPSecSPD_Handle_t* spdHandleArray;
} NPF_IPSecSPD_Handle_Array_t;
```

5.1.6 SPD Identity : NPF_IPSecSPD_Identity_t

```
/*
```

```
*     SPD Identity (ID and Handle)
*
*/
typedef struct
{
      NPF_IPSecSPD_ID_t spdID;
      NPF_IPSecSPD_Handle_t spdHandle;
      NPF_IPSecDirection_t spdDirection;
} NPF_IPSecSPD_Identity_t;
```

5.1.7 SPD Identity Array : NPF_IPSecSPD_Identity_Array_t

```
/*
*     SPD Identity Array
*
*/
typedef struct
{
      NPF_uint32_t nCount;
      NPF_IPSecSPD_Identity_t* spdIdentityArray;
} NPF_IPSecSPD_Identity_Array_t;
```

5.1.8 SPD Policy ID : NPF_IPSecPolicy_ID_t

```
/*
*     IPSec policy ID
*
*     Policy identity selected by the caller.
*/
typedef NPF_uint32_t NPF_IPSecPolicy_ID_t;
```

5.1.9 SPD Policy Handle : NPF_IPSecPolicy_Handle_t

```
/*
*     IPSec Policy Handle
*
*     Policy handle selected by the implementation
*/
typedef NPF_uint32_t NPF_IPSecPolicy_Handle_t;

/* The following values may not be used to indicate a valid policy handle */
#define NPF_IPSEC_POLICY_HANDLE_INVALID 0
```

5.1.10 SPD Policy Handle Array : NPF_IPSecPolicy_Handle_Array_t

```
/*
*     IPSec Policy Handle Array
*
*     Policy handles selected by the implementation
*/
typedef struct
{
      NPF_uint32_t nCount;
      NPF_IPSecPolicy_Handle_t * policyHandleArray;
} NPF_IPSecPolicy_Handle_Array_t;
```

5.1.11 SPD Policy Identity : NPF_IPSecPolicy_Identity_t

```
/*
*     Policy Identity (ID and Handle)
*
*/
typedef struct
{
      NPF_IPSecPolicy_ID_t policyID;
      NPF_IPSecPolicy_Handle_t policyHandle;
} NPF_IPSecPolicy_Identity_t;
```

5.1.12 SPD Policy Identity Array : NPF_IPSecPolicy_Identity_Array_t

```
/*
*     Policy Identity Array
*
*/
typedef struct
{
      NPF_uint32_t nCount;
      NPF_IPSecPolicy_Identity_t* policyIdentityArray;
} NPF_IPSecPolicy_Identity_Array_t;
```

5.1.13 SA ID : NPF_IPSecSA_ID_t

```
/*
*     IPSec SA Identity
*
*     SA identity selected by the caller.
*/
typedef NPF_uint32_t NPF_IPSecSA_ID_t;
```

5.1.14 SA Handle : NPF_IPSecSA_Handle_t

```
/*
*     IPSec SA Handle
*
*     SA handle selected by the implementation
*/
typedef NPF_uint32_t NPF_IPSecSA_Handle_t;

/* The following values may not be used to indicate a valid SA handle */
#define NPF_IPSEC_SA_HANDLE_INVALID 0
```

5.1.15 SA Identity : NPF_IPSecSA_Identity_t

```
/*
*     SA Identity (ID and Handle)
*
*/
typedef struct
{
      NPF_IPSecSA_ID_t policyID;
      NPF_IPSecSA_Handle_t SAHandle;
} NPF_IPSecSA_Identity_t;
```

5.1.16 SA Identity Array : NPF_IPSecSA_Identity_Array_t

```
/*
*     SA Identity Array
*
*/
typedef struct
{
      NPF_uint32_t nCount;
      NPF_IPSecSA_Identity_t* SAIdentityArray;
} NPF_IPSecSA_Identity_Array_t;
```

5.1.17 SPD Policy Action : NPF_IPSecPolicy_Action_t

```
/*
*     IPSec Policy Action
*     This is used to indicate the action taken when a given packet
*     matches an SPD rule.
*/
typedef enum
```

```
{
      NPF_IPSEC_POLICY_ACTION_IPSEC=1,    /* Apply IPSec */
      NPF_IPSEC_POLICY_ACTION_DISCARD=2,  /* Discard / Drop */
      NPF_IPSEC_POLICY_ACTION_BYPASS=3,   /* Allow to pass */
} NPF_IPSecPolicy_Action_t;
```

5.1.18 IP Network Address : NPF_IPSecPrefix_t

```
/*
*     IPSec IP Selector Network Address
*     Note: Basic IP types / structures are defined elsewhere.
*/
typedef union
{
      NPF_IPv4Prefix_t v4;
      NPF_IPv6Prefix_t v6;
} NPF_IPSecPrefix_t;
```

5.1.19 IP Address Range : NPF_IPSecRangeAddr_t

```
/*
*     IPSec IP Selector Range for IPv4 and IPv6
*/
typedef struct
{
      NPF_IPv4Address_t start;   /* First address in range */
      NPF_IPv4Address_t end;     /* Last address in range */
} NPF_IPSecIPv4RangeAddr_t;

typedef struct
{
      NPF_IPv6Address_t start;   /* First address in range */
      NPF_IPv6Address_t end;     /* Last address in range */
} NPF_IPSecIPv6RangeAddr_t;

/*
*     Consolidated range union for v4 and v6
*/
typedef union
{
      NPF_IPSecIPv4RangeAddr_t v4;
      NPF_IPSecIPv6RangeAddr_t v6;
} NPF_IPSecRangeAddr_t;
```

5.1.20 IP Selector Address Type : NPF_IPSecSelectorAddrType_t

```
/*
*     IPSec Selector Address type
*/
typedef enum
{
      NPF_IPSEC_ADDR_TYPE_SUBNET=0,
      NPF_IPSEC_ADDR_TYPE_RANGE=1
} NPF_IPSecSelectorAddrType_t;
```

5.1.21 IP Selector Address : NPF_IPSecSelectorAddr_t

```
/*
*     IPSec Selector Address
*/
typedef struct
{
```

```
      NPF_IPSecSelectorAddrType_t addrType;  /* subnet or range */
      union
      {
            NPF_IPSecPrefix_t       prefixAddr;
            NPF_IPSecRangeAddr_t    rangeAddr;
      } u;
} NPF_IPSecSelectorAddr_t;
```

5.1.22 IP Port Range: NPF_IPSecPortRange_t

```
/*
*     IPSec port range
*     Note: A specific port value (a point range) is defined
*     by setting start = end
*/
typedef struct
{
      NPF_uint16_t      start;      /* First port in range */
      NPF_uint16_t      end;        /* Last port in range */
} NPF_IPSecPortRange_t;
```

5.1.23 IPSec Selector : NPF_IPSecSelector_t

```
/*
*     IPSec Selector
*/

#define NPF_IPSEC_SELECTORRULEFLAG_CLEAR  (0) /* default */
#define NPF_IPSEC_SELECTORRULEFLAG_NOT_SET   (1 << 0)
#define NPF_IPSEC_SELECTORRULEFLAG_AND_SET   (1 << 1)

typedef struct
{
      NPF_uint8_t ruleFlags;              /* rule flag with values above */
      NPF_uint8_t IP_Version;             /* 4 = IPv4, 6 = IPv6 */
      NPF_uint8_t protocol;               /* IP Transp. Protocol or OPAQUE*/
      NPF_IPSecSelectorAddr_t srcIP;      /* Src Address */
      NPF_IPSecSelectorAddr_t dstIP;      /* Dst Address */
      NPF_IPSecPortRange_t srcPort;       /* IP Source Port or OPAQUE */
      NPF_IPSecPortRange_t dstPort;       /* IP Destination Port or OPAQUE */
} NPF_IPSecSelector_t;
```

There are two different ways to specify a specific IP address (either of `srcIP` or `dstIP`); one exploits a point range of addresses, using `NPF_IPSecRangeAddr_t`; the other exploits a network with maximal prefix length, using `NPF_IPSecPrefix_t`. Of these two ways, the second is preferred as it is less demanding for the implementation, especially in case of IPv6.

The selector ruleFlag is used to logically bind two sequential selectors in a selector array using the operator specified in the ruleFlag.

E.g. When adding a policy rule with two selectors (provided in an array), if the ruleFlag indicates an ‘AND’ operation in the first selector in the array, then this indicates that any 5-tuple matching this policy MUST match this selector and the following selector in the array, in order for the policy rule to be satisfied.

The default value of the ruleFlag will be 0 (zero), indicating that selectors in an array are bound in an ‘OR’ manner – i.e. a 5-tuple may match any selector in order to satisfy the rule.

Note** The ruleFlag is only meaningful when used in an array of selectors and furthermore is only applied to the next selector element in the array. It goes without saying that the ruleFlag in the final selector within a given array is ignored.

For compliance with this API, it is mandatory for an implementation to support the 'OR' operator only. Support for any other type of operation is optional. If an ruleFlag operation is not supported, it is permissible to return an error of type NPF_IPSEC_E_SELECTOR_RULEFLAG_UNSUPPORED

Additional Discussion

Currently there are only two bits of this flag needed to be defined to indicate the following:

- ***N* Bit (**`NPF_IPSEC_SELECTORRULEFLAG_NOT_SET`**)**: negated rule if set, otherwise positive rule.
- ***&* Bit (**`NPF_IPSEC_SELECTORRULEFLAG_AND_SET`**)**: RuleAND if set, otherwise RuleOR.

Thus, the default flag is as 0x00 for a Positive rule with OR condition.

Note that the key idea here is to associate the logic operation *AND* or *OR*, as well as the negation flag for *NOT,* to each selector rule, which define the evaluation behavior for each rule. The *RuleAND* and *RuleOR* flag of a selector defines the logical operation between the current selector and the next selector. That is, after the packet finishing one selector matching, the selector rule should decide the next evaluation behavior whether it could be,

- Continue to the next selector of the same policy (with grouping rules) to do the matching; or
- Continue to the first selector of the next policy (with another grouping rules) to do the matching; or
- Stop the next selector matching and apply the IPsec Action of the current policy.

The negated rules (i.e., selectors with **N Bit** set) express the meaning of *NOT* operation in the Boolean algebra. It excludes certain unwanted traffic to apply the policy action. For example, if the packet matches a negate rule and it is a *RuleAND,* the whole policy is considered as non-matching and the next policy with another rule group (in a selector list) is evaluated.

5.1.24 IPSec Generic IP Address

```
/*
*     Single IP address
*/
typedef union
{
      NPF_IPv4Address_t v4;               /* IPv4 address */
      NPF_IPv6Address_t v6;               /* IPv6 address */
} NPF_IPSecIPAddress_t;
```

5.1.25 Packet 5-tuple: NPF_IPSecPacketFields_t

```
/*
*     Packet 5-tuple
*/
typedef struct
{
      NPF_uint8_t IP_Version;             /* 4 = IPv4, 6 = IPv6 */
      NPF_uint8_t protocol;               /* IP Transp. Protocol or OPAQUE*/
      NPF_IPSecIPAddress_t srcIP;         /* Source address of packet */
      NPF IPSecIPAddress_t dstIP;         /* Destination address of packet */
      NPF_uint16_t srcPort;               /* IP Source Port or OPAQUE */
      NPF_uint16_t dstPort;               /* IP Destination Port or OPAQUE */
} NPF_IPSecPacketFields_t;
```

5.1.26 IPSec Tunnel Endpoint Address : NPF_IPSecTunEndAddr_t

```
/*
*     Tunnel Endpoint Addresses
*/
typedef struct
```

```
{
      NPF_uint8_t IP_Version;             /* 4 = IPv4, 6 = IPv6 */
      NPF_IPSecIPAddress_t srcIP;
      NPF_IPSecIPAddress_t dstIP;
} NPF_IPSecTunEndAddr_t;
```

5.1.27 IPSec Error Type : NPF_IPSecErrorType_t

```
/*
*     IPSec Error Type
*/
typedef NPF_uint32_t NPF_IPSecErrorType_t;  /* IPSec Error Type */
```

5.1.28 IPSec UDP Encapsulation Ports : NPF_IPSecUDP_Ports_t

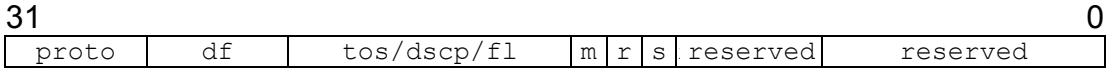
```
/*
*     IPSec UDP Encapsulation ports, if used
*/
typedef struct
{
      NPF_uint16_t srcPort;  /* IPSec UDP Src Port */
      NPF_uint16_t dstPort;  /* IPSec UDP Dst Port */
} NPF_IPSecUDP_Ports_t;
```

5.1.29 IPSec SA : NPF_IPSecSA_t

```
/*
*     IPSec SA Parameters
*/
typedef struct
{
    NPF_uint32_t  spi;        /* SPI / CPI for this SA */
    NPF_uint32_t  qos;        /* Explicit value for IPv6 dscp / Flow label */
                              /* May also be used for IPv4 TOS value */
                              /* dscp / TOS value is specified in the */
                              /* least significant 8 bits, flow label */
                              /* is specified in the next 20 bits */
    NPF_uint32_t  flags;            /* SA flags e.g. AH/ESP, TOS handling */
    NPF_uint8_t   authAlgo;         /* Authentication algorithm */
    NPF_uint8_t   encAlgo;          /* Encryption algorithm */
    NPF_uint32_t  softKbyteLimit;   /* Soft KB expiry */
    NPF_uint32_t  hardKbyteLimit;   /* Hard KB expiry */
    NPF_uint32_t  softSecsLimit;    /* Soft seconds expiry */
    NPF_uint32_t  hardSecsLimit;    /* Hard seconds expiry */
    NPF_IPSecTunEndAddr_t TE_Addr;  /* Tunnel endpoint src/dest addr */
    NPF_uint8_t   *authKey;         /* Authentication Key */
    NPF_uint16_t  authKeyLenBits;   /* Algorithm Key Length in bits */
    NPF_uint8_t   *encDecKey;       /* Enc / Dec key */
    NPF_uint16_t  encDecKeyLenBits; /* Algorithm Key Length in bits */
    NPF_uint8_t   replayWindowSize; /* Size of replay window in bytes */
} NPF_IPSecSA_t;
```

A number of comments apply to this definition:

- In case field `SPI` is used to hold a `CPI`, which is a 16 bit quantity, that CPI must be placed in the least significant bytes of `SPI`.
- Field `flags` is a bit vector. The semantics of individual subfields is given below where bits in the 32 bit integer are numbered from 31 (most significant bit) down to 0 (least significant bit).



`proto` is 4 bits wide and indicates the IPSec/compression header type employed by this SA. Acceptable values are:

```
NPF_IPSEC_PROTOCOL_ESP        /* RFC-2406 */
NPF_IPSEC_PROTOCOL_AH         /* RFC-2402 */
NPF_IPSEC_PROTOCOL_IPCOMP     /* IPCOMP */
NPF_IPSEC_PROTOCOL_V2_BIS     /* ESP / AH bis v2 */
```

`df` is 2 bits wide and specifies how to handle the don't fragment bit. This is only relevant for outbound SAs. The specific values are:

```
NPF_IPSEC_DF_COPY                   /* copy */
NPF_IPSEC_DF_CLEAR                  /* clear */
NPF_IPSEC_DF_SET                    /* set */
```

`tos/dscp/flowflabel` is 8 bits wide and specifies how to handle the type of service/DiffServ Code Point / flowlabel field. Specific values are:

```
NPF_IPSEC_QOS_TOS_COPY              /* copy */
NPF_IPSEC_QOS_TOS_CLEAR             /* clear */
NPF_IPSEC_QOS_TOS_SET               /* set */

NPF_IPSEC_QOS_DSCP_COPY             /* copy */
NPF_IPSEC_QOS_DSCP_CLEAR            /* clear */
NPF_IPSEC_QOS_DSCP_SET              /* set */

NPF_IPSEC_QOS_FLOWLABEL_COPY        /* copy */
NPF_IPSEC_QOS_FLOWLABEL_CLEAR       /* clear */
NPF_IPSEC_QOS_FLOWLABEL_SET         /* set */
```

`m` is 1 bit wide and indicates the mode of this SA. Specific values are:

```
NPF_IPSEC_SA_SAFLAGS_TUNNELMODE     /* Tunnel mode SA */
NPF_IPSEC_SA_SAFLAGS_TRANSPORTMODE  /* Transport mode SA */
```

s is 1 bit wide and indicates whether the NPU must control the lifetime in seconds for this SA. Specific values are:

```
NPF_IPSEC_SA_SAFLAGS_LIFESECS_ON  /* Lifetime (secs) monitoring on */
NPF_IPSEC_SA_SAFLAGS_LIFESECS_OFF /* Lifetime (secs) monitoring off */
```

`r` is 1 bit wide and indicates whether to validate sequence number replay. Specific values are:

```
NPF_IPSEC_SA_SAFLAGS_REPLAY_ON      /* Replay protection on */
NPF_IPSEC_SA_SAFLAGS_REPLAY_OFF     /* Replay protection off */
```

- Authentication and Encryption keys are supplied through pointers. The referenced key material must occupy a whole number of words and must be placed in consecutive bytes starting from the byte pointed to by the the relevant pointer. In case the value of the associated `KeyLenBits` is not divisible by 8, the final bits must be placed in the most significant bits of the last byte holding key material.

5.1.30 IPSec Policy: NPF_IPSecPolicy_t

```
/*
*     IPSec Policy
*/
typedef struct
{
      NPF_IPSecPolicy_ID_t          policyID;         /* Policy ID */
      NPF_IPSecPolicy_Action_t      policyAction;     /* Action */
```

```
      NPF_uint32_t                  selectorCount;    /* No of selectors */
      NPF_IPSecSelector_t           *selectorArray;   /* Selector array */
} NPF_IPSecPolicy_t;
```

5.1.31 Rate Limiting Events: NPF_IPSecEventLimit_t

```
/*
*     Rate Limiting Events
*/

typedef enum
{
      NPF_IPSEC_EVENT_LIMIT_TIME=1,       /* Time base limiting */
      NPF_IPSEC_EVENT_LIMIT_COUNT=2       /* Counter base limiting */
} NPF_IPSecEventLimitType_t;

typedef struct
{
      NPF_IPSecEvent_t              eventid;          /* Event ID */
      NPF_IPSecEventLimitType_t     limitType;        /* Limit type */
      union
      {
            NPF_uint32_t numPerSec; /* Event frequency in time */
            NPF_uint32_t nCount;    /* Generate 1 event for */
                                    /* every nCount encounters */

      }u;
} NPF_IPSecEventLimit_t;
```

5.2 Data Structures for Completion Callbacks

Just like many other NPF APIs, the IPSec API has just one callback function. This function corresponds to a large number of API calls. In most cases the response data are provided through a few base types, but for statistics more data is provided and this data is wrapped in structures.

5.2.1 IPSec SPD Policy Statistics : NPF_IPSecPolicy_Stats_t

```
/*
*     IPSec SPD Policy statistics
*/
typedef struct
{
    NPF_IPSecSPD_ID_t     spdID;    /* Identity of SPD containing policy */
    NPF_IPSecPolicy_ID_t  policyID; /* Identity of policy */
    NPF_uint32_t  policyErrors;     /* Policy validation (selector) errors */
    NPF_uint32_t  acquiredSAs;      /* Total number of SAs acquired */
    NPF_uint32_t  exceptions;       /* Total exceptions generated for SA */
} NPF_IPSecPolicy_Stats_t;
```

Field `policyErrors` has different semantics in the inbound and outbound directions. In the outbound direction it counts the number of packets that hit a policy, but for which no SA existed (yet). In the inbound direction it counts the number of packets that successfully passed decryption/authentication, but it was found that the packet didn't match a policy prescribing the applied protection.

5.2.2 IPSec SA Statistics : NPF_IPSecSA_Stats_t

```
/*
*     IPSec SA statistics
*/
typedef struct
```

```
{
    NPF_uint32_t  HMAC_Errors;       /* HMAC (hash) errors -inbound only- */
    NPF_uint32_t  decryptErrors;     /* Decryption  errors -inbound only- */
    NPF_uint32_t  replayErrors;      /* Replay attacks -inbound only- */
    NPF_uint32_t  selectorErrors;    /* SA selector validation errors */
    NPF_uint64_t  packetCount;       /* Packet count */
    /* The above counters may be reset by a reset_SA function call */
    NPF_uint64_t  bytesUsed;         /* # bytes the SA has been applied to */
    NPF_uint64_t  bytesRemaining;    /* # bytes remaining the SA may be
                                        applied to */
    NPF_uint32_t  secsUsed;          /* # secs used for this SA */
    NPF_uint32_t  secsRemaining;     /* # secs remaining before SA expires */
    NPF_uint64_t  sequenceNo;        /* Sequence number (Tx or Rx) */
    NPF_uint32_t  replayBitmap[MAX_SIZE_REPLAY_WINDOW];  /*  Copy of replay
                                       Bitmap -inbound only- */
} NPF_IPSecSA_Stats_t;

/*
*     IPSec SA Bundle Stats
*/
typedef struct
{
      NPF_IPSecSA_ID_t saID;        /* User identifier for this SA */
      NPF_uint32_t saCount;         /* Number of SAs in the bundle */
      NPF_IPSecSA_Stats_t* saBundleStats; /* Pointer to the SA bundle data */
} NPF_IPSecSA_BundleStats_t;
```

5.2.3 IPSec Callback Type : NPF_IPSecCallbackType_t

```
/*
* completion callback types
*/
typedef enum
{
      NPF_IPSEC_SPD_CREATE = 1,
      NPF_IPSEC_SPD_DESTROY = 2,
      NPF_IPSEC_SPD_BIND = 3,
      NPF_IPSEC_SPD_UNBIND = 4,
      NPF_IPSEC_SPD_FLUSH = 5,
      NPF_IPSEC_POLICY_CREATE = 6,
      NPF_IPSEC_POLICY_DESTROY = 7,
      NPF_IPSEC_POLICY_BATCH_CREATE = 8,
      NPF_IPSEC_POLICY_BATCH_DESTROY = 9,
      NPF_IPSEC_POLICY_BIND = 10,
      NPF_IPSEC_POLICY_UNBIND = 11,
      NPF_IPSEC_POLICY_BATCH_BIND = 12,
      NPF_IPSEC_POLICY_BATCH_UNBIND = 13,
      NPF_IPSEC_POLICY_CHANGEPRIORITY = 14,
      NPF_IPSEC_SA_ADD = 15,
      NPF_IPSEC_SA_REMOVE = 16,
      NPF_IPSEC_SA_RESERVESPI = 17,
      NPF_IPSEC_SA_RELEASESPI = 18,
      NPF_IPSEC_RATELIMIT_EVENTS = 19,
      NPF_IPSEC_POLICY_GET_STATS = 20,
      NPF_IPSEC_SA_GET_STATS = 21,
      NPF_IPSEC_QUERY_ALL_SPDS = 22,
      NPF_IPSEC_QUERY_ALL_SPD_BINDINGS = 23,
      NPF_IPSEC_QUERY_ALL_POLICIES = 24,
      NPF_IPSEC_QUERY_ALL_POLICY_BINDINGS = 25,
      NPF_IPSEC_QUERY_ALL_SAS = 26,
```

```
      NPF_IPSEC_QUERY_POLICY_DATA = 27,
      NPF_IPSEC_QUERY_SA_DATA = 28,
      NPF_IPSEC_QUERY_SPD_HANDLE = 29,
      NPF_IPSEC_QUERY_POLICY_HANDLE = 30,
      NPF_IPSEC_QUERY_SA_HANDLE = 31
} NPF_IPSecCallbackType_t;

#define IPSEC_MAX_CALLBACK_TYPES 31
```

5.2.4 IPSec Async Response : NPF_IPSecAsyncResponse_t

```
/*
* An asynchronous response contains an SPD handle,
* an error or success code, a direction indicator,
* and in most cases a function-specific type embedded
* in a union. One or more of these responses
* is passed to the callback function as an array
* within the NPF_IPSecCallbackData_t structure (below).
*/
typedef struct /* Asynchronous Response Structure */
{
      NPF_IPSecErrorType_t error;         /* Error code for this response */
      NPF_IPSecDirection_t direction;     /* if relevant */
      union /* Function-specific structures: */
      {
            /*
            *     The SPD Handle is returned for the following callbacks
            *
            *     NPF_IPSEC_SPD_CREATE
            *     NPF_IPSEC_SPD_BIND
            *     NPF_IPSEC_SPD_UNBIND
            *     NPF_IPSEC_SPD_FLUSH
            *     NPF_IPSEC_SPD_DESTROY
            *
            *     and optionally for the following callbacks, if the error
            *     code pertains to an invalid or duplicate SPD handle
            *
            *     NPF_IPSEC_POLICY_CREATE
            *     NPF_IPSEC_POLICY_BIND
            *     NPF_IPSEC_POLICY_UNBIND
            *     NPF_IPSEC_POLICY_DESTROY
            *
            *     NPF_IPSEC_POLICY_BATCH_CREATE
            *     NPF_IPSEC_POLICY_BATCH_DESTROY
            *     NPF_IPSEC_POLICY_BATCH_BIND
            *     NPF_IPSEC_POLICY_BATCH_UNBIND
            *
            *     NPF_IPSEC_SA_ADD
            *     NPF_IPSEC_SA_REMOVE
            *
            *     And for the following error codes when unbinding
            *     NPF_IPSEC_E_UNBIND_FAILED_LASTBINDING
            *     NPF_IPSEC_E_UNBIND_FAILED_BOUNDTOALL
            */
            NPF_IPSecSPD_Handle_t spdHandle;

            /*
            *     The policy handle is returned for the following callbacks
            *     NPF_IPSEC_POLICY_CREATE
            *     NPF_IPSEC_POLICY_BIND
            *     NPF_IPSEC_POLICY_UNBIND
            *     NPF_IPSEC_POLICY_CHANGEPRIORITY
            *     NPF_IPSEC_POLICY_DESTROY
```

```
*
*     NPF_IPSEC_POLICY_BATCH_DESTROY
*     NPF_IPSEC_POLICY_BATCH_BIND
*     NPF_IPSEC_POLICY_BATCH_UNBIND
*
*     and also for the following callbacks
*     NPF_IPSEC_SA_ADD or
*     NPF_IPSEC_SA_REMOVE,
*     NPF_IPSEC_SA_RESERVESPI,
*     NPF_IPSEC_SA_RELEASESPI, if the return code was
*     NPF_IPSEC_E_INVALID_POLICY_HANDLE or
*     NPF_IPSEC_E_DUPLICATE_POLICY_HANDLE
*     NPF_IPSEC_E_UNBIND_FAILED_LASTBINDING
*     NPF_IPSEC_E_UNBIND_FAILED_BOUNDTOALL
*/
NPF_IPSecPolicy_Handle_t policyHandle;

/*
*     The IPSec Selector is returned for a callback of
*     NPF_IPSEC_POLICY_CREATE
*     NPF_IPSEC_POLICY_BATCH_CREATE
*     with an error of
*     NPF_IPSEC_E_INVALID_SELECTOR. This structure will
*     contain a single selector, which was deemed invalid
*/
NPF_IN NPF_IPSecSelector_t    selector;

/*
*     The SA structure is returned in the NPF_IPSEC_SA_ADD
*     in the case the SA could not be added because
*     of some errors in the SA definition
*     or in the case of NPF_IPSEC_QUERY_SA_DATA
*/
NPF_IPSecSA_t SA;

/*
*     The SA Handle is returned when an SA is added,
*     NPF_IPSEC_SA_ADD, when the SA is removed,
*     NPF_IPSEC_SA_REMOVE
*/
NPF_IPSecSA_Handle_t    saHandle;

/*
*     The SPD policy counters structure is returned in the
*     NPF_IPSEC_POLICY_GET_STATISTICS call
*/
NPF_IPSecPolicy_Stats_t policyStats;

/*
*     The SA counters structure is returned in the
*     NPF_IPSEC_SA_GET_STATISTICS call
*/
NPF_IPSecSA_BundleStats_t saStats;

/*
* The SPI is returned in the NPF_IPSEC_RESERVE_SPI call
*/
```

```
NPF_uint32_t spi;

/* In the case of NPF_IPSEC_RATELIMIT_EVENTS, there is no
* additional information provided via this union
*/

/* A interface handle may be returned in the case
* the interface handle provided in the API was invalid
* NPF_IPSEC_E_INVALID_IF_HANDLE
*/
NPF_IfHandle_t ifHandle;

/*
* SPD ID may be returned for any function
* if the returned error is
* NPF_IPSEC_E_INVALID_SPD_ID
* NPF_IPSEC_E_SPD_ID_ALREADY_REGISTERED
*/
NPF_IPSecSPD_ID_t spdID;

/*
* Policy ID may be returned for any function
* if the returned error is
* NPF_IPSEC_E_INVALID_POLICY_ID
* NPF_IPSEC_E_POLICY_ID_ALREADY_REGISTERED
* NPF_IPSEC_E_DUPLICATE_POLICY_ID
*/
NPF_IPSecPolicy_ID_t policyID;

/*
* SA ID may be returned for any function
* if the returned error is
* NPF_IPSEC_E_INVALID_SA_ID
* NPF_IPSEC_E_SA_ID_ALREADY_REGISTERED
*/
NPF_IPSecSA_ID_t saID;

/*
* Batch function callbacks
* NPF_IPSEC_POLICY_BATCH_CREATE,
*/
NPF_IPSecPolicy_Handle_Array_t policyHandles;

/*
* NPF_IPSEC_QUERY_ALL_SPDS
* NPF_IPSEC_QUERY_ALL_POLICY_BINDINGS
* NPF_IPSEC_QUERY_SPD_HANDLE
*/
NPF_IPSecSPD_Identity_Array_t spdDataArray;

/*
* NPF_IPSEC_QUERY_ALL_SPD_BINDINGS
*/
NPF_IPSecInterface_Handle_Array_t interfaceDataArray;

/*
* NPF_IPSEC_QUERY_ALL_POLICIES
* NPF_IPSEC_QUERY_POLICY_HANDLE
*/
NPF_IPSecPolicy_Identity_Array_t policyDataArray;
```

```
            /*
            * NPF_IPSEC_QUERY_ALL_SAS
            * NPF_IPSEC_QUERY_SA_HANDLE
            */
            NPF_IPSecSA_Identity_Array_t saDataArray;

            /*
            * NPF_IPSEC_QUERY_POLICY_DATA
            */
            NPF_IPSecPolicy_t policyData;

      } u;
} NPF_IPSecAsyncResponse_t;
```

5.2.5 IPSec Callback Data : NPF_IPSecCallbackData_t

```
typedef struct
{
      NPF_IPSecCallbackType_t type;       /* Response to which function */
      NPF_boolean_t           allOK;      /* TRUE is all completed OK */
      NPF_uint32_t            nResp;      /* Number of responses in array */
      NPF_IPSecAsyncResponse_t *resp;     /* Pointer to response structures*/
} NPF_IPSecCallbackData_t;
```

5.3 Data Structures for Event Notifications

5.3.1 IPSec call handle : NPF_IPSecEventCallHandle_t

```
/*
*     IPSec call handle
*/
typedef NPF_uint32_t NPF_IPSecEventCallHandle_t; /* Event call handle */
```

5.3.2 IPSec Event : NPF_IPSecEvent_t

```
/*
*     IPSec Event
*/
typedef enum
{
      NPF_IPSEC_SA_ACQUIRE = 1,           /* SPD rule found but no SA */
      NPF_IPSEC_SA_EXPIRE_KB_SOFT = 2,    /* SA KB soft Expired */
      NPF_IPSEC_SA_EXPIRE_KB_HARD = 3,    /* SA KB hard Expired */
      NPF_IPSEC_SA_EXPIRE_SECS_SOFT = 4,  /* SA seconds soft Expired */
      NPF_IPSEC_SA_EXPIRE_SECS_HARD = 5,  /* SA seconds hard Expired */
      NPF_IPSEC_SA_PURGED = 6,            /* SA forcibly removed */
      NPF_IPSEC_SA_SEQUENCE_OVERFLOW = 7, /* Sequence # overflow */
      NPF_IPSEC_CLEARPACKET_DROPPED = 8,  /* Clear text Packet dropped */
      NPF_IPSEC_REPLAY_PACKET = 9,        /* Replay packet caught */
      NPF_IPSEC_WRONG_SPI = 10,           /* Unknown SPI-value in packet */
      NPF_IPSEC_AUTH_FAILED = 11,         /* Authentication failed */
      NPF_IPSEC_DECRYPTION_FAILED = 12,   /* IPSec decryption failed */
      NPF_IPSEC_INVALID_POLICY = 13,      /* Decryption-policy mismatch */
      NPF_IPSEC_POLICY_DISCARD = 14,      /* Discard policy */
      NPF_IPSEC_REASSEMBLY_REQD = 15,     /* Fragmented packet received */
      NPF_IPSEC_MEM_FULL = 16             /* A crypto NPU ran out of memory*/
} NPF_IPSecEvent_t;

#define IPSEC_MAX_EVENT_TYPES 16
```

The NPF_IPSEC_WRONG_SPI event occurs when the SA lookup for an arriving packet fails. This could happen because the arriving packet uses an SPI that is not currently used in the SAD.

5.3.3 IPSec event bitmap : NPF_IPSecEventMask_t

```
/*
*     IPSec event bitmask used in the event registration call.
*/
typedef NPF_uint32_t NPF_IPSecEventMask_t;

/*
* The following values can be set for the IPSecEventMask
*/

#define NPF_IPSEC_EVENT_ALL_DISABLE                    (0) /* disable all */
#define NPF_IPSEC_EVENT_SA_ACQUIRE_ENABLE              (1 << 0)
#define NPF_IPSEC_EVENT_SA_EXPIRE_KB_SOFT_ENABLE       (1 << 1)
#define NPF_IPSEC_EVENT_SA_EXPIRE_KB_HARD_ENABLE       (1 << 2)
#define NPF_IPSEC_EVENT_SA_EXPIRE_SECS_SOFT_ENABLE     (1 << 3)
#define NPF_IPSEC_EVENT_SA_EXPIRE_SECS_HARD_ENABLE     (1 << 4)
#define NPF_IPSEC_EVENT_SA_PURGED_ENABLE               (1 << 5)
#define NPF_IPSEC_EVENT_SA_SEQUENCE_OVERFLOW_ENABLE    (1 << 6)
#define NPF_IPSEC_EVENT_CLEARPACKET_DROPPED_ENABLE     (1 << 7)
#define NPF_IPSEC_EVENT_REPLAY_PACKET_ENABLE           (1 << 8)
#define NPF_IPSEC_EVENT_WRONG_SPI_ENABLE               (1 << 9)
#define NPF_IPSEC_EVENT_AUTH_FAILED_ENABLE             (1 << 10)
#define NPF_IPSEC_EVENT_DECRYPTION_FAILED_ENABLE       (1 << 11)
#define NPF_IPSEC_EVENT_INVALID_POLICY_ENABLE          (1 << 12)
#define NPF_IPSEC_EVENT_POLICY_DISCARD_ENABLE          (1 << 13)
#define NPF_IPSEC_EVENT_REASSEMBLY_REQD_ENABLE         (1 << 14)
#define NPF_IPSEC_EVENT_MEM_FULL_ENABLE                (1 << 15)

#define NPF_IPSEC_EVENT_ALL_ENABLE                     0xFFFFFFFF
```

5.3.3.1 Missing SA: NPF_IPSecSA_AcquireInfo_t

```
/*
*     Information about missing SA
*/
typedef struct
{
      NPF_IPSecSPD_ID_t spdID;              /* Client side SPD ID */
      NPF_IPSecPolicy_ID_t policyID;        /* Client side policy ID */
      NPF_IPSecPacketFields_t packet5Tuple; /* Fields that match selector */
      NPF_uint32_t acquireContext;          /* Acquire Context */
} NPF_IPSecSA_AcquireInfo_t;
```

5.3.3.2 Expired SA: NPF_IPSecSA_ExpireInfo_t

```
/*
*     SA Expired information used by NPF_IPSEC_SA_EXPIRE_KB_SOFT,
*     NPF_IPSEC_SA_EXPIRE_KB_HARD, NPF_IPSEC_SA_EXPIRE_SECS_SOFT,
*     NPF_IPSEC_SA_EXPIRE_SECS_HARD, NPF_IPSEC_SA_PURGED
*/
typedef struct
{
      NPF_IPSecSA_ID_t saID;              /* Client side ID for SA */
} NPF_IPSecSA_ExpireInfo_t;
```

5.3.3.3 Packet Dropped: NPF_IPSecPacketDropped_t

```
/*
*     Information about the dropped packet, should be used every time a
*     packet is dropped i.e. it shall be used together with
*     NPF_IPSEC_CLEARPACKET_DROPPED and
```

```
*     NPF_IPSEC_INVALID_POLICY
*     Depending on the event, one or more IDs must be provided. spdID and
*     policyID must be provided for NPF_IPSEC_CLEARPACKET_DROPPED and
*     NPF_IPSEC_INVALID_POLICY; saID must be provided for all events except
*     NPF_IPSEC_CLEARPACKET_DROPPED. If no value is required, any value will
*     serve as dummy value.
*/
typedef struct
{
      NPF_IPSecSPD_ID_t spdID;                  /* client side SPD ID */
      NPF_IPSecPolicy_ID_t policyID;            /* client side policy id */
      NPF_IPSecSA_ID_t saID;                    /* SA handle */
      NPF_IPSecPacketFields_t packet5Tuple;     /*offending packet 5-tuple */
} NPF_IPSecPacketDropped_t;
```

5.3.3.4 Crypto Packet Dropped: NPF_IPSecCryptoDropped_t

```
/*
*     Information about crypto packet dropped for events:
*     NPF_IPSEC_AUTH_FAILED
*     NPF_IPSEC_DECRYPTION_FAILED
*     NPF_IPSEC_SA_SEQUENCE_OVERFLOW
*     NPF_IPSEC_POLICY_DISCARD
*     NPF_IPSEC_REASSEMBLY_REQD
*/
typedef struct
{
      NPF_uint8_t IP_Version;             /* IPv4 = 4, IPv6 = 6 */
      NPF_IPSecIPAddress_t srcIP;         /* Src Address */
      NPF_IPSecIPAddress_t dstIP;         /* Dst Address */
      NPF_uint8_t protocol;               /* ESP = 50, AH = 51, COMP = 108 */
      NPF_uint32_t SPI;                   /* SPI value */
      NPF_uint32_t flowLabel;             /* IPv6 Flow label, 0 for IPv4 */
      NPF_uint8_t seqNo;                  /* Anti replay counter */
      NPF_uint8_t count;                  /* Repetition count */
} NPF_IPSecCryptoDropped_t;
```

This structure is used with two different events, `NPF_IPSEC_REPLAY_PACKET` and `NPF_IPSEC_WRONG_SPI`. The repetition count is used when rate limitation is in effect; it provides the number of packets that would have triggered the same event in case rate limitation were not in effect. When the repetition count is greater than one, field `seqNo` becomes ambiguous since it may vary between the offending packets; the rule that implementations should adhere to is to copy the sequence number from the last of the offending packets currently received.

5.3.4 IPSec Event Data : NPF_IPSecEventData_t

```
/*
*     IPSec Event Data
*/
typedef struct
{
      NPF_IPSecEvent_t eventType;
      union
      {
            NPF_IPSecSA_AcquireInfo_t     acquire;
            NPF_IPSecSA_ExpireInfo_t      expire;
            NPF_IPSecPacketDropped_t      packet;
            NPF_IPSecCryptoDropped_t      crypto;
      } event;
} NPF_IPSecEventData_t;
```

5.3.5 IPSec Event Array : NPF_IPSecEventArray_t

```
/*
*     IPSec Event Array
*/
typedef struct
{
      NPF_uint16_t            n_data;     /* Number of events in array */
      NPF_IPSecEventData_t    *eventDataArray;  /* Array of event
                                                   notifications */
} NPF_IPSecEventArray_t;
```

5.4 Error Codes

```
/*
*     Asynchronous error codes (returned in function callbacks)
*/

/*
*     IPSec reserved error codes in relation to other NPF APIs
*     Note** The maximum range is 100
*/
#define NPF_IPSEC_BASE_ERR 600 /* Base value of 600 wrt other NPF codes */
```

5.4.1 Generic Error Codes

```
/*
*     These are generic error codes, that can be returned in any callback
*/
#define NPF_IPSEC_GENERIC_ERROR_CODE_COUNT 20 /* Should be enough */

/* The Interface handle provided was not recognized as being valid */
#define NPF_IPSEC_E_INVALID_IF_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + 1)

/* Already registered */
#define NPF_IPSEC_E_ALREADY_REGISTERED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + 2)

/* Optional feature not supported */
#define NPF_IPSEC_E_OPTIONAL_FEATURE_NOT_SUPPORTED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + 3)

/* System was unable to allocate sufficient memory to complete operation */
#define NPF_IPSEC_E_NOMEMORY  ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR+4)
```

5.4.2 Specific Error Codes

```
/*
* Invalid IP Address error codes are defined in the IF Management API
* i.e.
*
* Invalid IP address : NPF_IF_E_INVALID_IPADDR
* Invalid IP net prefix length : NPF_IF_E_INVALID_NETPLEN
*/

/* Attempt to bind more than one SPD to an interface */
#define NPF_IPSEC_E_IF_ALREADY_BOUND \
      ((NPF_IPSecErrorType_t)NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 1)
```

```
/* Invalid SPD handle */
#define NPF_IPSEC_E_INVALID_SPD_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 2)

/* Invalid policy handle */
#define NPF_IPSEC_E_INVALID_POLICY_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 3)

/* Invalid SA handle */
#define NPF_IPSEC_E_INVALID_SA_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 4)

/* Invalid SPD ID - used in query functions */
#define NPF_IPSEC_E_INVALID_SPD_ID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 5)

/* Invalid policy ID - used in query functions */
#define NPF_IPSEC_E_INVALID_POLICY_ID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 6)

/* Invalid SA ID  - used in query functions */
#define NPF_IPSEC_E_INVALID_SA_ID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 7)

/* Invalid Policy Priority */
#define NPF_IPSEC_E_INVALID_POLICY_PRIORITY \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 8)

/* Invalid Policy Action */
#define NPF_IPSEC_E_INVALID_POLICY_ACTION \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 9)

/* Invalid selector */
#define NPF_IPSEC_E_INVALID_SELECTOR \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
      NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 10)

/* Invalid encryption algorithm, when attempting to add an SA */
#define NPF_IPSEC_E_INVALID_ENC_ALGO \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 11)

/* Invalid authentication algorithm, when attempting to add an SA */
#define NPF_IPSEC_E_INVALID_AUTH_ALGO \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 12)

/* Invalid compression algorithm, when attempting to add an SA */
#define NPF_IPSEC_E_INVALID_COMPRESS_ALGO \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 13)
```

```
/* NULL auth. and NULL encr. algorithms, when attempting to add an SA */
#define NPF_IPSEC_E_NULL_AUTH_NULL_ENC_ALGO \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 14)

/* Invalid encryption algorithm keylen, when attempting to add an SA */
#define NPF_IPSEC_E_INVALID_ENC_ALGO_KEYLEN \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 15)

/* Invalid authentication algorithm keylen, when attempting to add an SA */
#define NPF_IPSEC_E_INVALID_AUTH_ALGO_KEYLEN \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 16)

/* Cannot enable anti-replay when no authentication algorithm defined */
#define NPF_IPSEC_E_NO_REPLAY \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 17)

/* The SA SPI value provided is unacceptable to the implementation */
#define NPF_IPSEC_E_BAD_SPI \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 18)

/* ReserveSPI mode of operation not available */
#define NPF_IPSEC_E_RESERVESPIMODE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 19)

#define NPF_IPSEC_E_SPIINUSE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 20)

/* Duplicate IF handle specified */
#define NPF_IPSEC_E_DUPLICATE_IF_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 21)

/* Duplicate SPD handle specified */
#define NPF_IPSEC_E_DUPLICATE_SPD_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 22)

/* Duplicate policy handle specified */
#define NPF_IPSEC_E_DUPLICATE_POLICY_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 23)

/* Duplicate event ID specified */
#define NPF_IPSEC_E_DUPLICATE_EVENTID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 24)

/* Unrecognized event ID specified */
#define NPF_IPSEC_E_BAD_EVENTID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 25)
```

```
/* Failed to unbind an object */
#define NPF_IPSEC_E_UNBIND_FAILED_BOUNDTOALL \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 26)

/* Failed to unbind an object */
#define NPF_IPSEC_E_UNBIND_FAILED_LASTBINDING \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 27)

/* Failed to bind an object */
#define NPF_IPSEC_E_BIND_FAILED_BOUNDTOALL \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 28)

/* Failed to bind an object */
#define NPF_IPSEC_E_BIND_FAILED_ALREADYBOUND \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 29)

/* SPD ID already registered */
#define NPF_IPSEC_E_SPD_ID_ALREADY_REGISTERED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 30)

/* Policy ID already registered */
#define NPF_IPSEC_E_POLICY_ID_ALREADY_REGISTERED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 31)

/* SA ID already registered */
#define NPF_IPSEC_E_SA_ID_ALREADY_REGISTERED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 32)

/* Duplicate policy ID */
#define NPF_IPSEC_E_DUPLICATE_POLICY_ID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 33)

/* Duplicate selector */
#define NPF_IPSEC_E_DUPLICATE_SELECTOR \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
      NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 34)
/* Selector rule flag unsupported */
#define NPF_IPSEC_E_SELECTOR_RULEFLAG_UNSUPPORTED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
      NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 35)
```

5.5 Core Definitions

```
/*
* Core definitions
*/

/* IPSec Protocol */
#define NPF_IPSEC_PROTOCOL_ESP      0x01   /* RFC-2406 */
#define NPF_IPSEC_PROTOCOL_AH       0x02   /* RFC-2402 */
#define NPF_IPSEC_PROTOCOL_IPCOMP   0x04   /* IPCOMP */
```

```
#define NPF_IPSEC_PROTOCOL_V2_BIS   0x08   /* ESP / AH bis v2 */

/* DF Bit Handling */
#define NPF_IPSEC_DF_COPY           0x0   /* copy */
#define NPF_IPSEC_DF_CLEAR          0x1   /* clear */
#define NPF_IPSEC_DF_SET            0x2   /* set */

/* IPv4 TOS handling */
#define NPF_IPSEC_QOS_TOS_COPY      0x0   /* copy */
#define NPF_IPSEC_QOS_TOS_CLEAR     0x1   /* clear */
#define NPF_IPSEC_QOS_TOS_SET       0x2   /* set */

/* IPv6 DSCP / FLowlabel handling */
#define NPF_IPSEC_QOS_DSCP_COPY           0x00  /* copy */
#define NPF_IPSEC_QOS_DSCP_CLEAR          0x01  /* clear */
#define NPF_IPSEC_QOS_DSCP_SET            0x02  /* set */

#define NPF_IPSEC_QOS_FLOWLABEL_COPY      0x00  /* copy */
#define NPF_IPSEC_QOS_FLOWLABEL_CLEAR     0x10  /* clear */
#define NPF_IPSEC_QOS_FLOWLABEL_SET       0x20  /* set */

/* Tunnel / Transport Mode*/
#define NPF_IPSEC_SA_SAFLAGS_TUNNELMODE         0x0
#define NPF_IPSEC_SA_SAFLAGS_TRANSPORTMODE      0x1

/* Replay protection on / off */
#define NPF_IPSEC_SA_SAFLAGS_REPLAY_ON    0x0
#define NPF_IPSEC_SA_SAFLAGS_REPLAY_OFF   0x1

/* control lifetime in seconds */
#define NPF_IPSEC_SA_SAFLAGS_LIFESECS_ON  0x0
#define NPF_IPSEC_SA_SAFLAGS_LIFESECS_OFF 0x1

#define NPF_IPSEC_AALG_NONE           0
#define NPF_IPSEC_AALG_MD5HMAC        2
#define NPF_IPSEC_AALG_SHA1HMAC       3
#define NPF_IPSEC_AALG_AESXCBC        4

#define NPF_IPSEC_AALG_NONE_KEYBITS_LENGTH        0
#define NPF_IPSEC_AALG_MD5HMAC_KEYBITS_LENGTH     128
#define NPF_IPSEC_AALG_SHA1HMAC_KEYBITS_LENGTH    160
#define NPF_IPSEC_AALG_AESXCBC_KEYBITS_LENGTH     128

#define NPF_IPSEC_EALG_NONE           0
#define NPF_IPSEC_EALG_DESCBC         2
#define NPF_IPSEC_EALG_3DESCBC        3
#define NPF_IPSEC_EALG_NULL           11
#define NPF_IPSEC_EALG_AES            12

#define NPF_IPSEC_EALG_DESCBC_KEYBITS_LENGTH      64
#define NPF_IPSEC_EALG_3DESCBC_KEYBITS_LENGTH     192
#define NPF_IPSCE_EALG_NULL_KEYBITS_LENGTH        0
#define NPF_IPSEC_EALG_AES128_KEYBITS_LENGTH      128
#define NPF_IPSEC_EALG_AES192_KEYBITS_LENGTH      192
#define NPF_IPSEC_EALG_AES256_KEYBITS_LENGTH      256
```

6 Functions

6.1 Completion Callback

6.1.1 Completion Callback Function

Syntax

```
typedef void (*NPF_IPSecCallbackFunc_t)(
                  NPF_IN NPF_userContext_t           userContext,
                  NPF_IN NPF_correlator_t            correlator,
                  NPF_IN NPF_IPSecCallbackData_t     callbackData);
```

Description

The application registers this asynchronous response handling routine to the API implementation. The callback function is implemented by the application, and is registered to the API implementation through `NPF_IPSecRegister()` function.

Input Parameters

- `userContext`: The context item that was supplied by the application when the completion callback function was registered.
- `correlator`: The correlator item that was supplied by the application when the IPSec API function call was made. The correlator is used by the application mainly to distinguish between multiple invocations of the same function.
- `ipsecCallbackData`: Response information related to the IPSec API function call. Contains information that are common among all functions, as well as information that are specific to a particular function. See `NPF_IPSecCallbackData_t` definition for details.

Output Parameters

None.

Return Codes

None.

6.1.2 Completion Callback Registration Function

Syntax

```
NPF_error_t NPF_IPSecRegister(
                  NPF_IN  NPF_userContext_t          userContext,
                  NPF_IN  NPF_IPSecCallbackFunc_t    cbFunc,
                  NPF_OUT NPF_callbackHandle_t*      cbHandle);
```

Description

This function is used by the application to register its completion callback function for receiving asynchronous responses related to NPF ISPec API function calls. Applications MAY register multiple callback functions using this function. The callback function is identified by the pair of userContext and callbackFunc, and for each individual pair, a unique callback handle, cbHandle, will be assigned for future reference.

Since the callback function is identified by both userContext and callbackFunc, duplicate registration of the same callback function with a different userContext is allowed. Also, the same userContext can be shared among different callback functions. Duplicate registration of the same userContext and callbackFunc pair has no effect, and will output a handle that is already assigned to the pair, and will return NPF_IPSEC_E_ALREADY_REGISTERED.

Note: `NPF_IPSecRegister()` is a synchronous function and has no completion callback associated with it.

Input Parameters

- `userContext`: A context item for uniquely identifying the context of the application registering the completion callback function. The exact value will be provided back to the registered completion callback function as its 1st parameter when it is called. Application can assign any value to the userContext and the value is completely opaque to the NPF IPSec API implementation.
- `cbFunc`: A pointer to the completion callback function to be registered.

Output Parameters

- `cbHandle`: A unique identifier assigned for the registered `userContext` and `callbackFunc` pair. This handle will be used by the application to specify which callback function to be called when invoking asynchronous NPF IPSec API functions. It will also be used when de-registering the userContext and ipsecCallbackFunc pair.

Return Values

- `NPF_NO_ERROR`: The registration completed successfully.
- `NPF_E_BAD_CALLBACK_FUNCTION`: `callbackFunc` is `NULL`.
- `NPF_IPSEC_E_ALREADY_REGISTERED`: No new registration was made since the `userContext` and `callbackFunc` pair was already registered.

Note: Whether double registration should be treated as an error or not is dependent on the application.

6.1.3 Completion Callback Deregistration

Syntax

```
NPF_error_t NPF_IPSecDeregister(
      NPF_IN NPF_callbackHandle_t   cbHandle);
```

Description

This function is used by the application to de-register a pair of user context and callback function.

Note: If there are any outstanding calls related to the de-registered callback function, the callback function may be called for those outstanding calls even after de-registration.

Note: `NPF_IPSecDeregister()` is a synchronous function and has no completion callback associated with it.

Input Parameters

- `cbHandle`: The unique identifier representing the pair of user context and callback function to be de-registered.

Output Parameters

None.

Return Values

- `NPF_NO_ERROR`: The de-registration completed successfully.
- `NPF_E_BAD_CALLBACK_HANDLE`: The API implementation does not recognize the callback handle. There is no impact on the registered callback functions.

6.2 Event Notification

6.2.1 Event Handler Function

Syntax

```
typedef void (*NPF_IPSecEventCallFunc_t)(
      NPF_IN NPF_userContext_t      userContext,
      NPF_IN NPF_IPSecEventArray_t  eventArray);
```

Description

This handler function is for the application to register an event handling routine to the API implementation. One or more events can be notified to the application through a single invocation of this event handler function. Information on each event is represented in an array in the `ipsecEventArray` structure so that the application can traverse through the array and process each of the events. This event handler function is intended to be implemented by the application, and be registered to the API implementation through `NPF_IPSecEventRegister()` function.

Note: This function may be called any time after `NPF_IPSecEventRegister()` is called for it.

Input Parameters

- `userContext`: The context item that was supplied by the application when the event handler function was registered.
- `eventArray`: Data structure that contains an array of event information. See `NPF_IPSecEventArray_t` definition for details.

Output Parameters

None.

Return Codes

None.

6.2.2 Event Handler Registration Function

Syntax

```
NPF_error_t NPF_IPSecEventRegister(
      NPF_IN      NPF_userContext_t             userContext,
      NPF_IN      NPF_IPSecEventCallFunc_t      eventCallFunc,
      NPF_IN      NPF_IPSecEventMask_t          eventMask,
      NPF_OUT     NPF_IPSecEventCallHandle_t    *eventCallHandle);
```

Description

This function is used by an application to register its event handling routine for receiving notifications of IPSec events. Applications MAY register multiple event handling routines using this function. The event handling routine is identified by the pair of userContext and eventCallFunc, and for each individual pair, a unique eventCallHandle will be assigned for future reference.

Since the event handling routine is identified by both userContext and eventCallFunc, duplicate registration of the same event handling routine with a different userContext is allowed. Also, the same userContext can be shared among different event handling routines. Duplicate registration of the same userContext and eventCallFunc pair has no effect, and will output a handle that is already assigned to the pair, and will return NPF_IPSEC_E_ALREADY_REGISTERED.

This function also enables notifications for the events selected by the bits that are set in the eventMask parameter. A mask with all bits set selects all events of this SAPI. If the application wishes to change the selection of events, it may call the event registration function again with the same userContext and eventCallFunc, but with a different event selection mask. The events enabled are those whose bits were set in the most recent registration function call for a particular userContext and eventCallFunc pair.

Notes: Besides registering a handler function, this call enables events. The handler function could be called at any time following the invocation of `NPF_IPSecEventRegister()`. `NPF_IPSecEventRegister()` is a synchronous function and has no completion callback associated with it.

Special consideration needs to be taken when registering multiple applications for the `NPF_IPSEC_SA_ACQUIRE` event. It is the users responsibility to ensure that for any given acquire event, multiple bi-directional SAs are not added. This may happen if multiple applications are registered and each application initiates a separate key exchange (IKE session) for the same acquire event. Either a single application should be registered for this event or all applications registered for this event should synchronize in an out-of band manner to ensure that a single bi-directional SA is added in response to this event.

Input Parameters

- `userContext`: A context item for uniquely identifying the context of the application registering the event handler function. The exact value will be provided back to the registered event handler function as its 1st parameter when it is called. Application can assign any value to the userContext and the value is completely opaque to the API implementation.
- `eventCallFunc`: Pointer to the event handler function to be registered.
- `eventMask`: a bitmask defining the events to enable for this callback

Output Parameters

- eventCallHandle: A unique identifier assigned for the registered userContext and eventCallFunc pair. This handle will be used by the application de- registering the userContext and eventCallFunc pair.

Return Codes

- `NPF_NO_ERROR`: The registration completed successfully.
- `NPF_E_BAD_CALLBACK_HANDLE`: `eventCallFunc` is `NULL` or not recognized.
- `NPF_IPSEC_E_ALREADY_REGISTERED`: No new registration was made since the `userContext` and `eventCallFunc` pair was already registered.

- NPF_IPSEC_E_OPTIONAL_FEATURE_NOT_SUPPORTED: An attempt was made to leverage an optional feature within the API, which is not supported by this implementation. i.e. Some events are optional

6.2.3 Event Handler Deregistration Function

Syntax

```
NPF_error_t NPF_IPSecEventDeregister(
      NPF_IN  NPF_IPSecEventCallHandle_t  eventCallHandle);
```

Description

This function is used by an application to de-register a pair of user context and event handler function.

Input Parameters

- eventCallHandle: The unique identifier representing the pair of user context and event handler function to be de-registered.

Output Parameters

None.

Return Codes

- NPF_NO_ERROR: The de-registration completed successfully.
- NPF_E_BAD_CALLBACK_HANDLE: The API implementation does not recognize the event handler handle. There is no effect to the registered event handler functions.

6.3 Event definition signature

NPF IPSec implementations can generate the events listed in section 5.3.1, type NPF_IPSecEvent_t.

6.4 Completion Callbacks and Error Returns

Each of the functions defined in the IPSec API can return an immediate error, and each makes asynchronous callbacks. The error codes eligible for immediate return are those defined in [API Conventions]. They are:

- NPF_NO_ERROR: This value is returned when a function was successfully invoked.
- NPF_E_UNKNOWN: An unknown error occurred in the implementation such that there is no error code defined that is more appropriate or informative.
- NPF_BAD_CALLBACK_HANDLE: A function was invoked with a callback handle that did not correspond to a valid NPF callback handle as returned by a registration function, or a callback handle was registered with a registration function belonging to a different API than the function call where the handle was passed in.

All other error codes must be returned in an asynchronous callback response. They are defined with each function description.

6.5 Order of Operations

There are a few restrictions on the order of operations for the IPSec SAPI:

- `NPF_IPSecSPD_Create()` must precede any other IPSec SAPI operations.
- Creation of an interface (via Interface Management API) must proceed the `NPF_IPSecSPD_Bind()` operation, unless binding to all interfaces.

6.6 IPSec Service API

IPSec Service API Summary

The IPSec API includes the following functions and events:

SPD / Policy / SA Functions

- `NPF_IPSecSPD_Create()` creates an SPD and returns a handle.
- `NPF_IPSecSPD_Destroy()` unbinds and destroys a given SPD.
- `NPF_IPSecSPD_Bind()` binds a previously created SPD to one or more interfaces
- `NPF_IPSecSPD_UnBind()` unbinds an SPD from one or more interfaces
- `NPF_IPSecSPD_Flush()` removes all policy entries from an SPD.
- `NPF_IPSecPolicy_Create()` creates a policy and binds it to one or more SPDs. A unique handle is returned for the policy.
- `NPF_IPSecPolicy_Destroy()` Destroys a policy, performing an implicit unbind from all SPDs that it was bound to.
- `NPF_IPSecPolicy_BatchCreate()` creates a set of policies and binds them to single SPD. A unique handle is returned for each policy.
- `NPF_IPSecPolicy_BatchDestroy()` Destroys a set of policies, performing an implicit unbind from all SPDs that they were bound to.
- `NPF_IPSecPolicy_Bind()` binds a policy to one or more SPDs.
- `NPF_IPSecPolicy_UnBind()` unbinds a policy from one or more SPDs.
- `NPF_IPSecPolicy_BatchBind()` binds a set of policies to a given SPDs.
- `NPF_IPSecPolicy_BatchUnBind()` unbinds a set of policies from a given SPD.
- `NPF_IPSecPolicy_ChangePriority()` changes the priority of a previously defined policy.
- `NPF_IPSecSA_Add()` creates a security association (bundle); bindings between policies and SA (bundle) are established.
- `NPF_IPSecSA_Remove()` deletes a security association (bundle);

Statistics

- `NPF_IPSecPolicy_GetStats()` gets statistics for an SPD entry.
- `NPF_IPSecSA_GetStats()` gets statistics for an SA entry.

Inbound SPD Functions

- `NPF_IPSecInSA_ReserveSPI()` extracts an SA SPI from the underlying system, which is subsequently used in the addition of an inbound IPSec SA.
- `NPF_IPSecInSA_ReleaseSPI()` releases a SPI previously reserved by the NPF_IPSecInSA_ReserveSPI function, in the case the application is unable to add an SA.

Miscellaneous Helpers

- `NPF_IPSecRateLimitEvents()` Allows control over the number of events received at client.

Query Functions

- `NPF_IPSecQuery_AllSPDs()` Allows extraction of all SPD IDs and handles within the system.
- `NPF_IPSecQuery_AllSPDBindings()` Allows extraction of all interface handles that a given SPD is bound to.
- `NPF_IPSecQuery_AllPolicies()` Allows extraction of all policy IDs and handles bound to a given SPD.
- `NPF_IPSecQuery_AllPolicyBindings()` Allows extraction of all SPD IDs and handles that a given policy is bound to.
- `NPF_IPSecQuery_AllSAs()` Allows extraction of all SA IDs and handles bound to a given SPD.
- `NPF_IPSecQuery_PolicyData()` Allows extraction of policy data associated with a given policy handle.
- `NPF_IPSecQuery_SAData()` Allows extraction of SA data associated with a given SA handle.
- `NPF_IPSecQuery_SPDHandle()` Allows extraction of SPD handles, given a set of SPD IDs. Useful for 'lost' callbacks on creation / addition of an entry.
- `NPF_IPSecQuery_SAHandle()` Allows extraction of SA handles, given a set of SA IDs. Useful for 'lost' callbacks on creation / addition of an entry.
- `NPF_IPSecQuery_PolicyHandle()` Allows extraction of Policy handles, given a set of Policy IDs. Useful for 'lost' callbacks on creation / addition of an entry.

Event Definitions

- `NPF_IPSEC_SA_ACQUIRE`: A packet hit an outbound SPD, but no SA was found.
- `NPF_IPSEC_SA_EXPIRE_KB_SOFT`: The KB soft expiry limit has been reached.
- `NPF_IPSEC_SA_EXPIRE_KB_HARD`: The KB hard expiry limit has been reached.
- `NPF_IPSEC_SA_EXPIRE_SECS_SOFT`: The time soft expiry limit has been reached.
- `NPF_IPSEC_SA_EXPIRE_SECS_HARD`: The time hard expiry limit has been reached.
- `NPF_IPSEC_SA_PURGED`: The SA was removed due to user removing a policy or a rekey operation replaced this SA.
- `NPF_IPSEC_CLEARPACKET_DROPPED`: A packet was dropped due to no rule being found or because the packet was received in clear text while a matching policy required it to be protected by IPSec.
- `NPF_IPSEC_REPLAY_PACKET`: A packet previously processed was apparently received again.
- `NPF_IPSEC_WRONG_SPI`: A packet with an unknown SPI was seen.
- `NPF_IPSEC_AUTH_FAILED`: Authentication failure on a packet.
- `NPF_IPSEC_DECRYPTION_FAILED`: Decryption failure on a packet (e.g. due to invalid padding).
- `NPF_IPSEC_INVALID_POLICY`: A decrypted packet failed policy validation.
- `NPF_IPSEC_MEM_FULL`: Some NPU ran out of memory.
- `NPF_IPSEC_SA_SEQUENCE_OVERFLOW`: Sequence overflow in the given SA

- NPF_IPSEC_REASSEMBLY_REQD: A fragmented clear text packet was encountered after IPSec de-capsulation.

The following points should be observed when using the IPSec API.

- For any API call (excluding events) containing multiple values in an array, if an array element contains unacceptable values, then the function call will result in an error. Under these circumstances, even the (partial) valid data in the array will not be accepted.

6.6.1 Generic Description of IPSec Functions

Syntax

```
NPF_error_t NPF_IPSec<Noun><verb> (
      NPF_IN NPF_callbackHandle_t   cbHandle,
      NPF_IN NPF_correlator_t       cbCorrelator,
      NPF_IN NPF_errorReporting_t   errorReporting,
      NPF_IN NPF_IPSecDirection_t   direction,
      <…other function parameters…> );
```

Description

The IPSec function definitions follow the NPF framework guidelines.

Input Parameters

- cbHandle: the registered callback handle.
- cbCorrelator: the application's context for this call.
- errorReporting: the desired level of feedback.
- direction: direction of the policy / SA.
- plus any further parameters.

Output Parameters

Describe the output parameters for each function. In the IPSec SAPI there are no output parameters to any function.

Synchronous Return Codes

Only the following error codes are returned on making any function call. All other codes are returned in the asynchronous callback.

- NPF_NO_ERROR – The operation is in progress.
- NPF_E_UNKNOWN – The operation could not be completed due to problems encountered when handling the input parameters.
- NPF_E_BAD_CALLBACK_HANDLE – The callback handle is not valid.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_OPTIONAL_FEATURE_NOT_SUPPORTED: An attempt was made to leverage an optional feature within the API, which is not supported by this implementation. i.e. A given implementation may contain all the defined functional, but may return this error for the 'optional' functions.

Asynchronous response

Each of the asynchronous return codes are described in the appropriate functional section below, together with any data structures returned by the callback.

Discussion

Motivation for the function and its parameters, may be extended with a presentation of possible alternatives. Also may provide context information to clarify the usage.

6.6.2 NPF_IPSecSPD_Create : Create an SPD

Syntax

```
NPF_error_t NPF_IPSecSPD_Create (
      NPF_IN NPF_callbackHandle_t   cbHandle,
      NPF_IN NPF_correlator_t       cbCorrelator,
      NPF_IN NPF_errorReporting_t   errorReporting,
      NPF_IN NPF_IPSecDirection_t   direction,
      NPF_IN NPF_uint32_t           nifHandles,
      NPF_IN NPF_IfHandle_t         *ifHandleArray,
      NPF_IN NPF_IPSecSPD_ID_t      spdID);
```

Description

This function creates a new SPD, binds it to one or more interfaces and returns a unique handle associated with this SPD. The handle is used to perform any subsequent operations on this SPD.

An SPD must always be bound to at least one interface in order to avoid a case of constructing 'floating' SPDs.

On success, an implementation provided SPD handle is returned via the asynchronous callback. A handle value of NPF_IPSEC_SPD_HANDLE_INVALID is reserved and should not be returned as a valid handle.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `direction`: direction of the traffic this SPD is supposed to handle. Notice that the SPD handles returned for inbound and outbound directions will differ.
- `nifHandles`: number of interfaces to bind this SPD to. This parameter may be set to 0 (zero) in which case the SPD will be bound to all existing interfaces as well as to interfaces created in the future. Note** When binding an SPD in this manner, it is deemed global for life and it is not possible to unbind this SPD from a specific interface at a subsequent stage (or unbind the SPD from all interfaces). The only viable action is to destroy the SPD. If this is not desirable, then the SPD should be bound explicitly to all required interfaces.
- `ifHandleArray`: handles of `nifHandles` interfaces to which this SPD should be bound.
- `spdID`: identity associated with this SPD, as dictated by the client.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Additionally, the following may also be returned.

- `NPF_IPSEC_E_DUPLICATE_IF_HANDLE` – The same interface handle was provided more than once.

Asynchronous response

A callback of type `NPF_IPSEC_SPD_CREATE` is generated in response to this function call. This contains the return codes for any errors encountered. If the call is successful, then an SPD handle is returned. The SPD handle should be used in any subsequent interactions with the system. The following codes may be returned.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_IPSEC_E_DUPLICATE_IF_HANDLE` – The same interface handle was provided more than once.
- NPF_IPSEC_E_INVALID_IF_HANDLE – One or more of the interface handles provided was not recognized as being valid.
- NPF_IPSEC_E_SPD_ID_ALREADY_REGISTERED – The SPD ID given has already been used previously.

6.6.3 NPF_IPSecSPD_Destroy : Destroy an SPD

Syntax

```
NPF_error_t NPF_IPSecSPD_Destroy (
      NPF_IN NPF_callbackHandle_t   cbHandle,
      NPF_IN NPF_correlator_t       cbCorrelator,
      NPF_IN NPF_errorReporting_t   errorReporting,
      NPF_IN NPF_uint32_t           nSPDHandles,
      NPF_IN NPF_IPSecSPD_Handle_t  *spdHandleArray);
```

Description

This function administratively destroys one or more SPDs, which have been previously created by the `NPF_IPSecSPD_Create()` function. Before destroying the SPD, an implicit SPD Flush operation is performed in order to clean the SPD.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `nSPDHandles`: number of SPD handles to destroy. This parameter may be set to 0 (zero), in which case all previously created SPDs will be destroyed.
- `spdHandleArray`: pointer to an array of SPD handles.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Additionally, the following may also be returned.

- NPF_IPSEC_E_DUPLICATE_SPD_HANDLE – The same SPD handle was provided more than once.

Asynchronous response

A callback of type `NPF_IPSEC_SPD_DESTROY` is generated in response to this function call. This may contain up to nSPDHandles IPSec Async response structures, where each one contains one of the following error codes. Each returned structure corresponds to the received SPD handle from the spdHandleArray parameter above.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – The Received SPD handle was not recognized as being valid.
- `NPF_IPSEC_E_DUPLICATE_SPD_HANDLE` – The same SPD handle was provided more than once.

6.6.4 NPF_IPSecSPD_Bind : Bind an SPD to one or more interfaces

Syntax

```
NPF_error_t NPF_IPSecSPD_Bind (
      NPF_IN NPF_callbackHandle_t   cbHandle,
      NPF_IN NPF_correlator_t       cbCorrelator,
      NPF_IN NPF_errorReporting_t   errorReporting,
      NPF_IN NPF_IPSecSPD_Handle_t  spdHandle,
      NPF_IN NPF_uint32_t           nifHandles,
      NPF_IN NPF_IfHandle_t         *ifHandleArray);
```

Description

This function binds a previously created SPD to one or more interfaces using the interface handles. Once an SPD has been bound to an interface, all traffic entering / leaving that interface will first consult the associated SPD.

The interface handle(s) provided through this function must be valid handles that have been previously retrieved in creating the interface(s) using the interface management API.

If it is desirable to provide a global scope for the SPD, then the ‘number of interfaces’ parameter (nifHandles) may be set to 0. This is a special case implying that the SPD should be bound to all existing interfaces, as well as any interfaces created in the future. Note** Any subsequent attempts to bind this SPD to other explicit interfaces will fail with an error NPF_IPSEC_E_IF_ALREADY_BOUND. When binding an SPD to all interfaces in this manner, it is deemed global for life and it is not possible to unbind this SPD from a specific interface at a subsequent stage (or unbind the SPD from all interfaces). The only viable action is to destroy the SPD. If this is not desirable, then the SPD should be bound explicitly to all required interfaces.

This function is optional in the SAPI and is only required where dynamic binding must take place, generally some time after the creation of the SPD. It is envisioned that for most uses, static binding (provided at the time of creating an SPD) will suffice.

Note** An SPD may be bound to more then one interface, but an interface may not contain more then one SPD for any given direction. Additionally, binding an SPD to a well defined set of interfaces (instead of a single interface or ‘ALL’ interfaces) is an optional (not mandatory) feature.

Input Parameters

- `spdHandle`: the handle of a previously created SPD.

- `nifHandles`: number of interfaces to bind to. This parameter may be set to 0 (zero) in which case the SPD will be bound to all existing interfaces as well as to interfaces created in the future.
- `ifHandleArray`: handles of `nifHandles` interfaces to which this SPD should be bound.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions]. Additionally, the following may also be returned.

- `NPF_IPSEC_E_DUPLICATE_IF_HANDLE` – The same interface handle was provided more than once.

Asynchronous response

A callback of type `NPF_IPSEC_SPD_BIND` is generated in response to this function call. This contains the return codes for any errors encountered. If the call is successful, then an `NPF_NO_ERROR` value is returned. The following codes may be returned.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_IF_HANDLE – One or more of the interface handles provided was not recognized as being valid.
- NPF_IPSEC_E_IF_ALREADY_BOUND – One or more of the interfaces is already protected by an SPD.
- `NPF_IPSEC_E_DUPLICATE_IF_HANDLE` – The same interface handle was provided more than once.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – The Received SPD handle was not recognized as being valid.

Discussion

Parameters `nifHandles` and `ifHandleArray` could be replaced by a single parameter that indicates a single interface. Then an SPD could be bound to just a single interface and thus be more in line with the strict wording of [RFC 2401]. The proposed two parameters, however, make it possible to formulate a catch-all SPD and also paves the way for conserving memory by sharing data structures for the SPD across interfaces.

6.6.5 NPF_IPSecSPD_UnBind : UnBind an SPD from one or more interfaces

Syntax

```
NPF_error_t NPF_IPSecSPD_UnBind (
      NPF_IN NPF_callbackHandle_t   cbHandle,
      NPF_IN NPF_correlator_t       cbCorrelator,
      NPF_IN NPF_errorReporting_t   errorReporting,
      NPF_IN NPF_IPSecSPD_Handle_t  spdHandle,
      NPF_IN NPF_uint32_t           nifHandles,
      NPF_IN NPF_IfHandle_t         *ifHandleArray);
```

Description

This function unbinds an SPD from one or more interfaces using the interface handles. Once an SPD has been unbound from an interface, all traffic entering / leaving that interface will NOT consult the associated SPD.

The interface handle(s) provided through this function must be valid handles that have been previously retrieved in creating the interface(s) using the interface management API and used to bind with the SPD.

At any given time, an SPD must be bound to at least one interface. This is to avoid a case of having 'floating' SPDs in the system. An attempt to unbind an SPD from the last interface that it is bound to will result in an error.

Note** An SPD may be bound to more then one interface, but an interface may not contain more then one SPD in any given direction. Additionally, binding an SPD to a well defined set of interfaces (instead of a single interface or 'ALL' interfaces) is an optional (not mandatory) feature.

Input Parameters

- `spdHandle`: the handle of a previously created SPD.
- `nifHandles`: number of interfaces to unbind from. This parameter may NOT be set to 0 (zero), to avoid having 'floating' SPDs.
- `ifHandleArray`: handles of `nifHandles` interfaces from which this SPD should be unbound.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Additionally, the following may also be returned.

- `NPF_IPSEC_E_DUPLICATE_IF_HANDLE` – The same interface handle was provided more than once.

Asynchronous response

A callback of type `NPF_IPSEC_SPD_UNBIND` is generated in response to this function call. This contains the return codes for any errors encountered. If the call is successful, then an `NPF_NO_ERROR` value is returned. The following codes may be returned.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_IF_HANDLE – One or more of the interface handles provided was not recognized as being valid.
- `NPF_IPSEC_E_DUPLICATE_IF_HANDLE` – The same interface handle was provided more than once.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – The Received SPD handle was not recognized as being valid.
- NPF_IPSEC_E_UNBIND_FAILED_LASTBINDING – The operation failed to complete; generally this is a result of an attempt to unbind the SPD from the last interface that it is bound to.
- NPF_IPSEC_E_UNBIND_FAILED_BOUNDTOALL – The operation failed to complete; generally this is a result of the SPD being bound to all interfaces.

6.6.6 NPF_IPSecSPD_Flush : Flush an SPD

Syntax

```
NPF_error_t NPF_IPSecSPD_Flush (
      NPF_IN NPF_callbackHandle_t   cbHandle,
      NPF_IN NPF_correlator_t       cbCorrelator,
      NPF_IN NPF_errorReporting_t   errorReporting,
      NPF_IN NPF_uint32_t           nSPDHandles,
      NPF_IN NPF_IPSecSPD_Handle_t  *spdHandleArray);
```

Description

This function administratively clears one or more SPDs, which have been previously created by the `NPF_IPSecSPD_Create()` function. All entries within the SPD will be cleared. Since a policy may be shared among several SPDs, clearing an SPD does not necessarily imply that the policies vanish. However, should the operation remove the final reference to a policy, then the policy must be removed along with all SAs associated with the policy. This should not give rise to any events being sent upwards through the API. The SPD itself will remain until the SPD destroy function is called.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `nSPDHandles`: number of SPD handles to clear. This parameter may be set to 0 (zero), in which case all previously created SPDs will be flushed.
- `spdHandleArray`: pointer to an array of SPD handles.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Additionally, the following may also be returned.

- `NPF_IPSEC_E_DUPLICATE_SPD_HANDLE` – The same SPD handle was provided more than once.

Asynchronous response

A callback of type `NPF_IPSEC_SPD_FLUSH` is generated in response to this function call. This may contain up to nSPDHandles IPSec Async response structures, where each one contains one of the following error codes. Each returned structure corresponds to the received SPD handle from the spdHandleArray parameter above.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_IPSEC_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_IPSEC_E_INVALID_SPD_HANDLE` – The Received SPD handle was not recognized as being valid.
- `NPF_IPSEC_E_DUPLICATE_SPD_HANDLE` – The same SPD handle was provided more than once.

6.6.7 NPF_IPSecPolicy_Create : Create a policy and add to one or more SPDs

Syntax

```
NPF_error_t NPF_IPSecPolicy_Create (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_t            *policy,
      NPF_IN NPF_uint32_t                 nHandles,
      NPF_IN NPF_IPSecSPD_Handle_t        *spdHandleArray,
      NPF_IN NPF_IPSecDirection_t         direction,
      NPF_IN NPF_uint32_t                 *prioritiesArray);
```

Description

This function creates a single policy and adds this to one or more SPDs which have been previously created by the `NPF_IPSecSPD_Create()` function. A single policy handle is returned in an asynchronous callback, if successful.

A policy must be bound to at least one SPD in order to avoid a case of 'floating' policies in the system. This policy handle will be a globally unique policy identifier. The policy may contain one or more selectors (5-tuples) and an associated action. The action will be applied to all selectors defined in the policy.

A handle value of NPF_IPSEC_POLICY_HANDLE_INVALID is reserved and should not be returned as a valid handle.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `policy`: pointer to a structure containing the IPSec policy data
- `nHandles`: number of SPD handles into which the policy is to be added. This parameter may be set to 0 (zero), implying a global policy for all SPDs. Note** When binding a policy to all SPDs in this manner, it is deemed global for life and it is not possible to unbind this policy from a specific SPD at a subsequent stage (or unbind the policy from all SPDs). The only viable action is to destroy the policy. If this is not desirable, then the policy should be bound explicitly to all required SPDs.
- `spdHandleArray`: pointer to an array of SPD handles or NULL, if nHandles is 0.
- `direction`: This is only pertinent, if the nHandles is set to zero. In other cases, the direction may be ignored and implicitly derived from the SPD direction.
- `prioritiesArray`: array of numeric value indicating priority associated with a given SPD that the policy is being added to. This value is used for sorting SPD policies by the implementation. The number of elements in the array should be equal to the nHandles parameter.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Additionally, the following may also be returned.

- `NPF_IPSEC_E_DUPLICATE_SPD_HANDLE` – The same SPD handle was provided more than once.

- NPF_IPSEC_E_DUPLICATE_SELECTOR – The given selector(s) are duplicated in the array.

Asynchronous response

A callback of type NPF_IPSEC_POLICY_CREATE is generated in response to this function call. This may contain a single IPSecAsyncResponse_t structure containing one of the following error codes.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – One of the given SPD handles was not recognized as being valid.
- NPF_IPSEC_E_INVALID_POLICY_ACTION – The given policy action was not recognized as being valid.
- NPF_IPSEC_E_INVALID_SELECTOR – The given selector(s) is invalid.
- NPF_IPSEC_E_DUPLICATE_SELECTOR – The given selector(s) are duplicated in the array.
- NPF_IPSEC_E_DUPLICATE_SPD_HANDLE – The same SPD handle was provided more than once.
- NPF_IPSEC_E_POLICY_ID_ALREADY_REGISTERED – The policy ID given has already been used previously
- NPF_IPSEC_E_SELECTOR_RULEFLAG_UNSUPPORTED – Rule flag within the selector is not supported by this implementation.

6.6.8 NPF_IPSecPolicy_Destroy: Delete a policy from all SPDs

Syntax

```
NPF_error_t NPF_IPSecPolicy_Destroy (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle);
```

Description

This function removes a single policy from all SPDs into which it has been previously added by the NPF_IPSecPolicy_Create() function. When a policy is removed, all (inbound and outbound) SAs associated with that policy must be removed. The removal of these SAs may give rise to events.

Input Parameters

- cbHandle: the registered callback handle.
- cbCorrelator: the application's context for this call.
- errorReporting: the desired level of feedback.
- policyHandle: handle of the policy to delete.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_POLICY_DESTROY` is generated in response to this function call. This may contain a single `IPSecAsyncResponse_t` structure, containing one of the following error codes.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_IPSEC_E_INVALID_POLICY_HANDLE` – The given policy handle was not recognized as being valid.

6.6.9 NPF_IPSecPolicy_BatchCreate : Create a set of policies and add to a single SPD

Syntax

```
NPF_error_t NPF_IPSecPolicy_BatchCreate (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_uint32_t                 nPolicies,
      NPF_IN NPF_IPSecPolicy_t            *policyArray,
      NPF_IN NPF_IPSecSPD_Handle_t        spdHandle,
      NPF_IN NPF_IPSecDirection_t         direction,
      NPF_IN NPF_uint32_t                 *prioritiesArray);
```

Description

This function creates a number of policies and adds them to a single SPDs which has been previously created by the `NPF_IPSecSPD_Create()` function. An array of policy handles is returned in an asynchronous callback, if successful.

This policy handles will be globally unique policy identifiers. The policy may contain one or more selectors (5-tuples) and an associated action. The action will be applied to all selectors defined in each policy. If it is desirable to bind policies to all SPDs, then the spdHandle value may be zero (0).

The return handle value of NPF_IPSEC_POLICY_HANDLE_INVALID is reserved and should not be returned as a valid policy handle.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `nPolicies`: number of policies contained in the policyArray. This parameter may NOT be 0 (zero).
- `policyArray`: pointer to an array of policy structures containing the IPSec policy data
- `spdHandle`: Handle of the SPD to which the policies are being added. This value may be 0 (zero), indicating that the policies are being added to all SPDs in the system for a given direction. Note** When binding a policy to all SPDs in this manner, it is deemed global for life and it is not possible to unbind this policy from a specific SPD at a subsequent stage (or unbind the policy from all SPDs). The only viable action is to destroy the policy. If this is not desirable, then the policy should be bound explicitly to all required SPDs.

- `direction`: This is only pertinent, if the spdHandle is set to zero. In other cases, the direction may be ignored and implicitly derived from the SPD direction.
- `prioritiesArray`: array of numeric value indicating priority associated with a given policy within the SPD. This value is used for sorting SPD policies by the implementation. The number of elements in the array should be equal to the nPolicies parameter.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Additionally, the following may also be returned.

- `NPF_IPSEC_E_DUPLICATE_SPD_HANDLE` – The same SPD handle was provided more than once.
- `NPF_IPSEC_E_DUPLICATE_SELECTOR` – The given selector(s) are duplicated in the array.
- `NPF_IPSEC_E_SELECTOR_RULEFLAG_UNSUPPORTED – Rule flag within the selector is not supported by this implementation.`

Asynchronous response

A callback of type `NPF_IPSEC_POLICY_BATCH_CREATE` is generated in response to this function call. This may contain a single `IPSecAsyncResponse_t` structure containing one of the following error codes.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – The given SPD handle was not recognized as being valid.
- `NPF_IPSEC_E_INVALID_POLICY_ACTION` – The given policy action was not recognized as being valid.
- `NPF_IPSEC_E_INVALID_SELECTOR` – One of the given selector(s) is invalid.
- `NPF_IPSEC_E_DUPLICATE_SELECTOR` – The given selector(s) are duplicated in the array.
- NPF_IPSEC_E_POLICY_ID_ALREADY_REGISTERED – The policy ID given has already been used previously.
- NPF_IPSEC_E_DUPLICATE_POLICY_ID – One of the policy IDs is duplicated in the policyArray.
- `NPF_IPSEC_E_SELECTOR_RULEFLAG_UNSUPPORTED – Rule flag within the selector is not supported by this implementation.`

6.6.10 NPF_IPSecPolicy_BatchDestroy: Delete a set of policies from a single SPD

Syntax

```
NPF_error_t NPF_IPSecPolicy_BatchDestroy (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_uint32_t                 nPolicies,
      NPF_IN NPF_IPSecPolicy_Handle_t     *policyHandleArray);
```

Description

This function removes a set of policies from the SPDs into which they have been previously added by the `NPF_IPSecPolicy_Create()` or `NPF_IPSecPolicy_BatchCreate()` functions. When a policy is removed, all (inbound and outbound) SAs associated with the policies must be removed. The removal of these SAs may give rise to events.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `policyHandle`: handle of the policy to delete.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions] and the following.

- NPF_IPSEC_E_DUPLICATE_POLICY_HANDLE – One of the policy handles is duplicated in the policyHandleArray.

Asynchronous response

A callback of type `NPF_IPSEC_POLICY_BATCH_DESTROY` is generated in response to this function call. This may contain a single `IPSecAsyncResponse_t` structure, containing one of the following error codes.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_IPSEC_E_INVALID_POLICY_HANDLE` – One of the policy handles was not recognized as being valid.
- NPF_IPSEC_E_DUPLICATE_POLICY_HANDLE – One of the policy handles is duplicated in the policyHandleArray.

6.6.11 NPF_IPSecPolicy_Bind : Bind a policy to one or more SPDs

Syntax

```
NPF_error_t NPF_IPSecPolicy_Bind (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
      NPF_IN NPF_uint32_t                 nHandles,
      NPF_IN NPF_IPSecSPD_Handle_t        *spdHandleArray,
      NPF_IN NPF_IPSecDirection_t         direction,
      NPF_IN NPF_uint32_t                 *prioritiesArray);
```

Description

This function binds a single policy into a one or more SPDs which have been previously created by the `NPF_IPSecSPD_Create()` function. A single policy handle is returned in an asynchronous

callback, if successful. If the policy is to be bound to all SPDs in the system, then a value of 0 should be provided for the nHandles parameter.

This does not change any previous bindings to any SPDs, but merely provides the delta set of handles for the new bindings.

This function is optional in the SAPI and is only required where dynamic binding must take place, generally, some time after the creation of a policy. It is envisioned that for most uses, static binding (provided at the time of creating an SPD) will suffice.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `policyHandle`: handle of a previous created policy
- `nHandles`: number of SPD handles into which the policy is to be bound. This parameter may be set to 0 (zero), implying a global policy for all SPDs in a given direction. Note** When binding a policy to all SPDs in this manner, it is deemed global for life and it is not possible to unbind this policy from a specific SPD at a subsequent stage (or unbind the policy from all SPDs). The only viable action is to destroy the policy. If this is not desirable, then the policy should be bound explicitly to all required SPDs.
- `spdHandleArray`: pointer to an array of SPD handles or NULL, if nHandles is 0.
- `direction`: This is only pertinent, if the spdHandle is set to zero. In other cases, the direction may be ignored and implicitly derived from the SPD direction.
- `prioritiesArray`: array of numeric value indicating priority associated with a given SPD that the policy is being added to. This value is used for sorting SPD policies by the implementation. The number of elements in the array should be equal to the nHandles parameter.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Additionally, the following may also be returned.

- `NPF_IPSEC_E_DUPLICATE_SPD_HANDLE` – The same SPD handle was provided more than once.

Asynchronous response

A callback of type `NPF_IPSEC_POLICY_BIND` is generated in response to this function call. This may contain a single `IPSecAsyncResponse_t` structure containing one of the following error codes.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – One of the given SPD handles was not recognized as being valid.
- `NPF_IPSEC_E_DUPLICATE_SPD_HANDLE` – The same SPD handle was provided more than once.

- NPF_IPSEC_E_INVALID_POLICY_HANDLE – The given policy handle was not recognized as being valid.
- NPF_IPSEC_E_BIND_FAILED_BOUNDTOALL – The given policy handle is already bound to all SPDs
- NPF_IPSEC_E_BIND_FAILED_ALREADYBOUND – The given policy handle is already bound to the given SPD

6.6.12 NPF_IPSecPolicy_UnBind : Unbind a policy from one or more SPDs

Syntax

```
NPF_error_t NPF_IPSecPolicy_UnBind (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
      NPF_IN NPF_uint32_t                 nSPDHandles,
      NPF_IN NPF_IPSecSPD_Handle_t        *spdHandleArray);
```

Description

This function unbinds a single policy from one or more SPDs which have been previously created by the NPF_IPSecSPD_Create() function. A single policy handle is returned in an asynchronous callback, if successful.

At any given time, a policy must be bound to at least one SPD. This is to avoid a case of having 'floating' policies in the system. An attempt to unbind a policy from the last SPD that it is bound to will result in an error.

Input Parameters

- cbHandle: the registered callback handle.
- cbCorrelator: the application's context for this call.
- errorReporting: the desired level of feedback.
- policyHandle: handle of a previous created policy
- nSPDHandles: number of SPD handles from which the policy is to be unbound. This must NOT be 0 (zero), as unbinding from all SPDs will leave 'floating' policies.
- spdHandleArray: pointer to an array of SPD handles.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Additionally, the following may also be returned.

- NPF_IPSEC_E_DUPLICATE_SPD_HANDLE – The same SPD handle was provided more than once.

Asynchronous response

A callback of type NPF_IPSEC_POLICY_UNBIND is generated in response to this function call. This may contain a single IPSecAsyncResponse_t structure containing one of the following error codes.

- NPF_NO_ERROR – The operation was successful.

- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – One of the given SPD handles was not recognized as being valid.
- `NPF_IPSEC_E_DUPLICATE_SPD_HANDLE` – The same SPD handle was provided more than once.
- `NPF_IPSEC_E_INVALID_POLICY_HANDLE` – The given policy handle was not recognized as being valid.
- NPF_IPSEC_E_UNBIND_FAILED_LASTBINDING – The operation failed to complete; generally this is a result of an attempt to unbind the policy from the last SPD that it is bound to.
- NPF_IPSEC_E_UNBIND_FAILED_BOUNDTOALL – The operation failed to complete; generally this is a result of the policy being bound to all SPDs.

6.6.13 NPF_IPSecPolicy_BatchBind : Bind a set of policies to a single SPD

Syntax

```
NPF_error_t NPF_IPSecPolicy_BatchBind (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_uint32_t                 nPolicies,
      NPF_IN NPF_IPSecPolicy_Handle_t     *policyHandleArray,
      NPF_IN NPF_IPSecSPD_Handle_t        spdHandle,
      NPF_IN NPF_IPSecDirection_t         direction,
      NPF_IN NPF_uint32_t                 *prioritiesArray);
```

Description

This function binds a set of policies into a single SPD which has been previously created by the `NPF_IPSecSPD_Create()` function. An array of policy handles are returned in an asynchronous callback, if successful. If the policy is to be bound to all SPDs in the system, then a value of 0 should be provided for the spdHandle parameter.

This does not change any previous bindings to the SPD.

This function is optional in the SAPI and is only required where dynamic binding must take place, generally, some time after the creation of a policy. It is envisioned that for most uses, static binding (provided at the time of creating an SPD) will suffice.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `nPolicies`: number of policy handles contained in the policyHandleArray parameter. This parameter may NOT be 0 (zero).
- `policyHandleArray`: pointer to an array of policy handles.
- `spdHandle`: Handle of the SPD to which the policies are being bound. This value may be 0 (zero), indicating that the policies are being bound to all SPDs in the system for a given direction. Note** When binding a policy to all SPDs in this manner, it is deemed global for life and it is not possible to unbind this policy from a specific SPD at a subsequent stage (or

unbind the policy from all SPDs). The only viable action is to destroy the policy. If this is not desirable, then the policy should be bound explicitly to all required SPDs.

- `direction`: This is only pertinent, if the spdHandle is set to zero. In other cases, the direction may be ignored and implicitly derived from the SPD direction.
- `prioritiesArray`: array of numeric value indicating priority associated with a given policy within the SPD. This value is used for sorting SPD policies by the implementation. The number of elements in the array should be equal to the nPolicies parameter.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Additionally, the following may also be returned.

- `NPF_IPSEC_E_DUPLICATE_POLICY_HANDLE` – The same policy handle was provided more than once.

Asynchronous response

A callback of type `NPF_IPSEC_POLICY_BATCH_BIND` is generated in response to this function call. This may contain a single `IPSecAsyncResponse_t` structure containing one of the following error codes.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – The given SPD handle was not recognized as being valid.
- `NPF_IPSEC_E_INVALID_POLICY_HANDLE` – One of the given policy handles was not recognized as being valid.
- NPF_IPSEC_E_POLICY_ID_ALREADY_REGISTERED – The policy ID given has already been used previously.
- NPF_IPSEC_E_DUPLICATE_POLICY_ID – One of the policy IDs is duplicated in the policyHandleArray.
- `NPF_IPSEC_E_BIND_FAILED_BOUNDTOALL` – The given policy handle is already bound to all SPDs
- `NPF_IPSEC_E_BIND_FAILED_ALREADYBOUND` – The given policy handle is already bound to the given SPD

6.6.14 NPF_IPSecPolicy_BatchUnBind : Unbind a set of policies from a given SPD

Syntax

```
NPF_error_t NPF_IPSecPolicy_BatchUnBind (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_uint32_t                 nPolicies,
      NPF_IN NPF_IPSecPolicy_Handle_t     *policyHandleArray,
      NPF_IN NPF_IPSecSPD_Handle_t        spdHandle);
```

Description

This function unbinds a set of policies from a given SPD which have been previously created by the `NPF_IPSecSPD_Create()` or `NPF_IPSecSPD_Bind()` functions or the equivalent 'batch' functions. An array of policy handles is returned in an asynchronous callback, if successful.

At any given time, a policy must be bound to at least one SPD. This is to avoid a case of having 'floating' policies in the system. An attempt to unbind a policy from the last SPD that it is bound to will result in an error.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `nPolicies`: number of policies contained in the policyArray. This parameter may NOT be 0 (zero).
- `policyArray`: pointer to an array of policy structures containing the IPSec policy data
- `spdHandle`: Handle of the SPD from which the policies are being unbound. This value may NOT be 0 (zero), indicating that the policies are being unbound from all SPDs in the system, as this would cause 'floating' policies.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Additionally, the following may also be returned.

- `NPF_IPSEC_E_DUPLICATE_POLICY_HANDLE` – The same policy handle was provided more than once.

Asynchronous response

A callback of type `NPF_IPSEC_POLICY_BATCH_UNBIND` is generated in response to this function call. This may contain a single `IPSecAsyncResponse_t` structure containing one of the following error codes.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – One of the given SPD handles was not recognized as being valid.
- NPF_IPSEC_E_INVALID_POLICY_HANDLE – One of the given policy handles was not recognized as being valid.
- `NPF_IPSEC_E_DUPLICATE_POLICY_HANDLE` – The same policy handle was provided more than once.
- NPF_IPSEC_E_UNBIND_FAILED_LASTBINDING – The operation failed to complete; generally this is a result of an attempt to unbind a policy from the last SPD that it is bound to.
- NPF_IPSEC_E_UNBIND_FAILED_BOUNDTOALL – The operation failed to complete; generally this is a result of the policy being bound to all SPDs.

- NPF_IPSEC_E_DUPLICATE_POLICY_ID – One of the policy IDs is duplicated in the policyHandleArray.

6.6.15 NPF_IPSecPolicy_ChangePriority: Change the priority of a Policy

Syntax

```
NPF_error_t NPF_IPSecPolicy_ChangePriority (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecSPD_Handle_t        spdHandle,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
      NPF_IN NPF_uint32_t                 newPriority);
```

Description

This function changes the priority of a previously defined policy.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `spdHandle`: handle of the SPD containing the policy
- `policyHandle`: handle identifying this policy
- `priority`: numeric value indicating the new priority associated with this policy. This value is used for sorting SPD policies by the implementation.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_POLICY_CHANGEPRIORITY` is generated in response to this function call. This may contain a single `IPSecAsyncResponse_t` structure containing one of the following error codes.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_POLICY_HANDLE – The policy handle provided was unrecognized.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – The Received SPD handle was not recognized as being valid.

6.6.16 NPF_IPSecSA_Add : Add an SA into the SAD

Syntax

```
NPF_error_t NPF_IPSecSA_Add (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
```

```
NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
NPF_IN NPF_IPSecSA_ID_t             saID,
NPF_IN NPF_uint32_t                 nSA_Count,
NPF_IN NPF_IPSecSA_t                *saArray,
NPF_IN NPF_IPSecUDP_Ports_t         udpPorts,
NPF_IN NPF_uint32_t                 selectorCount,
NPF_IN NPF_IPSecSelector_t          *selectorArray,
NPF_IN NPF_IPSecSA_Handle_t         previousSA,
NPF_IN NPF_IPSecDirection_t         direction,
NPF_IN NPF_uint32_t                 acquireContext,
NPF_IN NPF_uint32_t                 vendorSpecHint);
```

Description

This function adds a single SA (bundle) and associates this with the given policy and hence, implicitly, with the SPD(s) and interface(s) associated with the policy. The SA contains all information needed in order to process an inbound or outbound packet, depending on the SA direction. Notice that the direction of the SA need not be supplied, as it will be deduced from the policy, and hence implicitly the SPD, it is being bound to. If the SA is added successfully, an implementation generated SA handle is returned to the caller in the asynchronous callback. This handle is guaranteed to be unique within the scope of this policy.

A handle value of NPF_IPSEC_SA_HANDLE_INVALID is reserved and should not be returned as a valid handle.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `policyHandle`: handle of the policy with which the SA (bundle) is to be associated.
- `saID`: ID of SA from callers perspective.
- `nSA_Count`: number of SAs passed. This is only pertinent to SAs used in transport adjacency and does not support any other type of nesting. The maximum number of SAs should be 3 (AH + ESP + IPCOMP).
- `saArray`: pointer to an array of SAs. This may contain 1 or more SAs. Multiple SAs may be present when using transport adjacency nesting (any combination of ESP / AH / IPCOMP).
- `udpPorts`: structure indicating the UDP ports to use if UDP encapsulation. If ports are zero, then UDP encapsulation is not used.
- `selectorCount`: number of selectors (5-tuples) associated with this SA (bundle).
- `selectorArray`: pointer to the selector associated with innermost SA (all other SAs will be nested based on SA relationships and there is no need for selectors for those).
- `previousSA`: The handle of a previous SA, if this SA is replacing an existing SA (i.e. rekey). A value of zero (0) indicates that this is the first SA in the system for the policy handle and selector(s) provided. The previous SA handle should be reflective of the SA direction (inbound / outbound). i.e. When adding an outbound SA, the previousSA handle should be the outbound SA handle and when adding an inbound SA, this handle should be the inbound SA handle. Note* As a new SA is added to replace an existing SA, it allows an implementation to remove the previous SA (SA being replaced). This is further discussed in the discussion section below.
- `direction`: direction of this SA (inbound / outbound)

- `acquireContext`: acquire context that was passed into an Acquire Event via the `NPF_IPSecSA_AcquireInfo_t` structure. This should only be passed down if this SA is being added as a result of an Acquire event. Otherwise, this value is 0.
- `vendorSpecHint`: implementation specific hint, which may be interpreted according to proprietary vendor extensions (i.e. Traffic Spec, Explicit nesting hints, etc.)

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions] and the following.

- `NPF_IPSEC_E_DUPLICATE_SELECTOR` – The given selector(s) are duplicated in the array.
- `NPF_IPSEC_E_SELECTOR_RULEFLAG_UNSUPPORTED - Rule flag within the selector is not supported by this implementation.`

Asynchronous response

A callback of type `NPF_IPSEC_SA_ADD` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing an implementation defined SA handle associated with this SA bundle, if the SA is successfully added. This SA handle can be subsequently used to delete the SA. On failure to add the SA, the following error codes may be returned.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_IPSEC_E_INVALID_POLICY_HANDLE` – The given policy handle was not recognized as being valid.
- `NPF_IPSEC_E_INVALID_SELECTOR` – The given selector(s) is invalid.
- `NPF_IPSEC_E_DUPLICATE_SELECTOR` – The given selector(s) are duplicated in the array.
- `NPF_IPSEC_E_INVALID_ENC_ALGO` – The encryption algorithm provided was not recognized by the system as being valid.
- `NPF_IPSEC_E_INVALID_AUTH_ALGO` – The authentication algorithm provided was not recognized by the system as being valid.
- `NPF_IPSEC_E_INVALID_COMPRESS_ALGO` – The compression algorithm provided was not recognized by the system as being valid.
- `NPF_IPSEC_E_NULL_AUTH_NULL_ENC_ALGO` – Attempt to employ NULL authentication and NULL encryption together in ESP.
- `NPF_IPSEC_E_INVALID_ENC_ALGO_KEYLEN` – The encryption algorithm key length provided did not match the provided encryption algorithm.
- `NPF_IPSEC_E_INVALID_AUTH_ALGO_KEYLEN` – The authentication algorithm key length provided did not match the provided authentication algorithm.
- `NPF_IPSEC_E_NO_REPLAY` – SA indicated using the anti-replay protection, but the authentication service / algorithm was not enabled.
- `NPF_IPSEC_E_BAD_SPI` – The value of the SPI provided within the SA was not acceptable to the implementation. This error is returned if the implementation detects that an attempt is made to add an SA, without first calling the NPF_IPSecInSA_ReserveSPI function, if in a previous operation having used that mode of operation.

- `NPF_IPSEC_E_SA_ID_ALREADY_REGISTERED` – The given SA ID has already been used in a previous call.
- `NPF_IPSEC_E_SELECTOR_RULEFLAG_UNSUPPORTED – Rule flag within the selector is not supported by this implementation.`

Discussion

It is worth observing the behavior if an SA is added when there is an existing SA for the same policy in the system. This should be treated as a 'rekeyed' SA from an IPSec / IKE perspective. When this condition is encountered, the new SA can be immediately used for outbound traffic, whereas the previous inbound SA should be kept for a period of time, t, to cater for the reception of any packets using this previous SA or until an inbound packet using the new SA is seen. The value of the grace period, t, is implementation dependent. An SA hard expiry event must be generated if the KB hard limit is reached or the grace period expires, whichever comes first. If the SA is removed due to a hard expiry, then a NPF_IPSEC_SA_EXPIRE_HARD event must be generated. If the SA expires due to a rekey, then a NPF_IPSEC_SA_PURGED event must be generated. Additionally, if the user application explicitly wishes to remove the previous SA, it can do so using the NPF_IPSecSA_Remove() function below.

6.6.17 NPF_IPSecSA_Remove: Remove an SA from the SAD

Syntax

```
NPF_error_t NPF_IPSecSA_Remove (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
      NPF_IN NPF_IPSecSA_Handle_t         saHandle);
```

Description

This function removes a single SA, or SA bundle, identified by the SA handle.

Input Parameters

- `cbHandle`: the registered callback handle.
- cbCorrelator: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `policyHandle`: handle of the policy with which this SA is associated.
- `saHandle`: handle of the SA to delete.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_SA_REMOVE` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing an error code indicating if the SA was successfully deleted. On failure to delete the SA, the following error codes may be returned.

- NPF_NO_ERROR – The operation was successful.

- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_IPSEC_E_INVALID_POLICY_HANDLE` – The given policy handle is null or invalid.
- `NPF_IPSEC_E_INVALID_SA_HANDLE`: The received handle of the SA was not recognized as being valid.

6.6.18 NPF_IPSecInSA_ReserveSPI : Reserve and return an inbound SPI from the system

Syntax

```
NPF_error_t NPF_IPSecInSA_ReserveSPI (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
      NPF_IN NPF_uint32_t                 selectorCount,
      NPF_IN NPF_IPSecSelector_t          *selectorArray);
```

Description

This function may be used in certain optimized scenarios, where the inbound SPI is dictated by the implementation. This allows the implementation to use the SPI as some form of a direct index lookup for the inbound SA, instead of using the traditional 3-tuple (destination address, protocol, SPI) as a key to search a database.

Note** The concurrent use of this function in conjunction with adding an SA without using this function first is implementation dependent. If a given implementation of this API is not able to support concurrency, it should return an error based on which method is supported (or even based on the first method used dictates the user choice for all subsequent calls).

The function extracts a single SPI from the underlying system, which is subsequently used in addition on an inbound SA. The SPI is returned in the asynchronous callback `NPF_IPSEC_RESERVE_SPI or an appropriate error code is returned`.

In the case of transport adjacency bundles, the SPI should only be extracted for the outermost inbound SA. It is up to the user application to select a SPI / CPI for the inner SAs.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `policyHandle`: handle of the policy for which the SPI is being extracted.
- `selectorCount`: number of selectors (5-tuples) associated with the SA for which the SPI is being extracted.
- `selectorArray`: pointer to the selector array containing all the selectors associated with the SA.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions] and the following.

- `NPF_IPSEC_E_DUPLICATE_SELECTOR` – The given selector(s) are duplicated in the array.

- NPF_IPSEC_E_SELECTOR_RULEFLAG_UNSUPPORTED – Rule flag within the selector is not supported by this implementation.

Asynchronous response

A callback of type NPF_IPSEC_SA_RESERVESPI is generated in response to this function call. A single asynchronous response, NPF_IPSecAsyncResponse_t, will be passed to the callback function containing an error code indicating if operation was successfully. The following error codes may be returned.

- NPF_NO_ERROR: Operation successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_POLICY_HANDLE – The given policy handle is null or invalid.
- NPF_IPSEC_E_INVALID_SELECTOR – The given selector(s) is invalid.
- NPF_IPSEC_E_DUPLICATE_SELECTOR – The given selector(s) are duplicated in the array.
- NPF_IPSEC_E_NOSPI: The system was unable to return a SPI for some reason (i.e. Database full).
- NPF_IPSEC_E_RESERVESPIMODE: Returned if an SA has previously been added without a call to get SPI and the implementation does not support NPF_IPSecInSA_ReserveSPI concurrent usage.
- NPF_IPSEC_E_SELECTOR_RULEFLAG_UNSUPPORTED – Rule flag within the selector is not supported by this implementation.

6.6.19 NPF_IPSecInSA_ReleaseSPI : Release an inbound SPI from the system

Syntax

```
NPF_error_t NPF_IPSecInSA_ReleaseSPI (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
      NPF_IN NPF_uint32_t                 spi);
```

Description

This function is used to release a SPI, previously allocated with a NPF_IPSecInSA_ReserveSPI () functional call, ONLY if the user application has not successfully added an SA associated with that SPI. This function is only to be used in error conditions, where after extracting a SPI, the user application finds that it is unable to complete the SA negotiation for some reason.

Input Parameters

- cbHandle: the registered callback handle.
- cbCorrelator: the application's context for this call.
- errorReporting: the desired level of feedback.
- policyHandle: handle of the policy for which the SPI is being extracted.
- spi: the value of the SPI that is being released.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_SA_RELEASESPI` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing an error code indicating if operation was successfully. The following error codes may be returned.

- `NPF_NO_ERROR`: Operation successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_IPSEC_E_INVALID_POLICY_HANDLE` – The given policy handle is null or invalid.
- `NPF_IPSEC_E_BAD_SPI` – The value of the SPI being released is unrecognized
- `NPF_IPSEC_E_SPIINUSE`: The system was unable to release the SPI, as it is already being used by some inbound SA.
- `NPF_IPSEC_E_RESERVESPIMODE`: Returned if an SA has previously been added without a call to get SPI and the implementation does not support NPF_IPSecInSA_ReserveSPI / NPF_IPSecInSA_ReleaseSPI concurrent usage

6.6.20 NPF_IPSecRateLimitEvents: Control event frequency

Syntax

```
NPF_error_t NPF_IPSecRateLimitEvents (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_uint32_t                 countEventData,
      NPF_IN NPF_IPSecEventLimit_t        *eventLimitArray);
```

Description

This function allows control over the number of events generated for each event type. Rate limiting may be set based on time or the accumulation of multiple events of the same type into a single event to the client application.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `countEventData`: the number of events being configured
- `eventLimitArray`: rate limiting data associated with each event

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions] and the following.

- NPF_IPSEC_E_DUPLICATE_EVENTID – The event ID was duplicated in the event array

Asynchronous response

A callback of type `NPF_IPSEC_RATELIMIT_EVENTS` is generated in response to this function call. The following status codes may be returned.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_DUPLICATE_EVENTID – The event ID was duplicated in the event array
- NPF_IPSEC_E_BAD_EVENTID – The event ID specified was not recognized as being valid.

Discussion

There are different ways to specify rate limitation; the one provided above is just one out of many. It could also be considered whether more discriminating rate limitation should be specifiable, e.g. per SPI, SPD, policy etc. These options could alternatively be provided through proprietary extensions.

6.6.21 NPF_IPSecPolicy_GetStats : Extract dynamic statistics for a given Policy

Syntax

```
NPF_error_t NPF_IPSecPolicy_GetStats (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
      NPF_IN NPF_boolean_t                resetStats);
```

Description

This function extracts statistics for a given policy and optionally resets the counters after extraction. The statistics will be returned for this single SPD via the asynchronous callback function.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `policyHandle`: handle of the policy for which stats are being retrieved.
- `resetStats`: value of TRUE indicates all policy stats should be reset to zero, after extraction.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_POLICY_GETSTATS` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback

function containing the requested statistics using the SPD policy statistics structure, `NPF_IPSecPolicy_Stats_t`.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_IPSEC_E_INVALID_POLICY_HANDLE` – The given policy handle was not recognized as being valid.

6.6.22 NPF_IPSecSA_GetStats : Extract dynamic statistics for a given SA

Syntax

```
NPF_error_t NPF_IPSecSA_GetStats (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
      NPF_IN NPF_IPSecSA_Handle_t         saHandle,
      NPF_IN NPF_boolean_t                resetStats);
```

Description

This function extracts statistics for a given SA and optionally resets some of the counters after extraction. The statistics will be returned for a single SA via the asynchronous callback function.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `policyHandle`: handle of policy containing this SA.
- `saHandle`: handle of SA for which statistics are needed.
- `resetStats`: value of TRUE indicates dynamic stats should be reset to zero, after extraction. To determine which stats are reset, please consult the SA statistics structure.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_SA_GETSTATS` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing the requested statistics using the SA statistics structure, `NPF_IPSecSA_BundleStats_t`.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_IPSEC_E_INVALID_POLICY_HANDLE` – The given policy handle was not recognized as being valid.

- NPF_IPSEC_E_INVALID_SA_HANDLE – The given SA handle was not recognized as being valid.

6.6.23 NPF_IPSecQuery_AllSPDs: Extract all SPD IDs and associated handles

Syntax

```
NPF_error_t NPF_IPSecQuery_AllSPDs (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting);
```

Description

Extraction of all SPD IDs, handles and associated SPD direction stored in the system.

Input Parameters

- cbHandle: the registered callback handle.
- cbCorrelator: the application's context for this call.
- errorReporting: the desired level of feedback.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type NPF_IPSEC_QUERY_ALL_SPDS is generated in response to this function call. A single asynchronous response, NPF_IPSecAsyncResponse_t, will be passed to the callback function containing the requested information.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.

6.6.24 NPF_IPSecQuery_AllSPDBindings: Extract all interface handles for a given SPD

Syntax

```
NPF_error_t NPF_IPSecQuery_AllSPDBindings (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecSPD_Handle_t        spdHandle);
```

Description

Extraction of all interface handles that a given SPD is bound to.

Input Parameters

- cbHandle: the registered callback handle.
- cbCorrelator: the application's context for this call.
- errorReporting: the desired level of feedback.
- spdHandle: the handle of the SPD being queried.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_QUERY_ALL_SPD_BINDINGS` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing the requested information.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – The given SPD handle was not recognized as being valid.

6.6.25 NPF_IPSecQuery_AllPolicies: Extract all policy IDs and handles for a given SPD

Syntax

```
NPF_error_t NPF_IPSecQuery_AllPolicies (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecSPD_Handle_t        spdHandle);
```

Description

Extraction of all policy IDs and handles for a given SPD.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- spdHandle: the handle of the SPD being queried.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_QUERY_ALL_POLICIES` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing the requested information.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.

- NPF_IPSEC_E_INVALID_SPD_HANDLE – The given SPD handle was not recognized as being valid.

6.6.26 NPF_IPSecQuery_AllPolicyBindings: Extract all SPDs for a given policy

Syntax

```
NPF_error_t NPF_IPSecQuery_AllPolicyBindings (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle);
```

Description

Extraction of all SPD handles for a given policy.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- policyHandle: the handle of the policy being queried.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_QUERY_ALL_POLICY_BINDINGS` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing the requested information.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_POLICY_HANDLE – The given policy handle was not recognized as being valid.

6.6.27 NPF_IPSecQuery_AllSAs: Extract all SA IDs and handles for a given SPD

Syntax

```
NPF_error_t NPF_IPSecQuery_AllSAs (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecSPD_Handle_t        spdHandle);
```

Description

Extraction of all SA IDs and handles for a given SPD.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- spdHandle: the handle of the SPD being queried.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_QUERY_ALL_SAS` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing the requested information.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_SPD_HANDLE – The given SPD handle was not recognized as being valid.

6.6.28 NPF_IPSecQuery_PolicyData: Extract policy data for a given policy handle

Syntax

```
NPF_error_t NPF_IPSecQuery_PolicyData (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle);
```

Description

Extraction of policy data for a given policy handle.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- policyHandle: the handle of the policy being queried.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_QUERY_POLICY_DATA` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing the requested information.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_POLICY_HANDLE – The given policy handle was not recognized as being valid.

6.6.29 NPF_IPSecQuery_SAData: Extract SA data for a given SA handle

Syntax

```
NPF_error_t NPF_IPSecQuery_SAData (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecSA_Handle_t         saHandle);
```

Description

Extraction SA data associated with a given SA handle.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- saHandle: the handle of the SA being queried.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_QUERY_SA_DATA` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing the requested information.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_SA_HANDLE – The given SA handle was not recognized as being valid.

6.6.30 NPF_IPSecQuery_SPDHandle : Extract SPD handles based on SPD IDs

Syntax

```
NPF_error_t NPF_IPSecQuery_SPDHandle (
```

```
    NPF_IN NPF_callbackHandle_t         cbHandle,
    NPF_IN NPF_correlator_t             cbCorrelator,
    NPF_IN NPF_errorReporting_t         errorReporting,
    NPF_IN_NPF_uint32_t                 countSPDIDs,
    NPF_IN NPF_IPSecSPD_ID_t            *spdIDArray);
```

Description

Allows extraction of the internal handles associated with a set of user identifiers for that object. The handles are returned via the associated callback using `NPF_IPSecSPD_Identity_Array_t`.

Note** If the SPD handle returned for any element in the array is NPF_IPSEC_SPD_HANDLE_INVALID, this indicates that the associated SPD ID was not recognized as being valid.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `countSPDIDs`: the number of Ids in the ID Array
- `spdIDArray`: array of user defined SPD Ids

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_QUERY_SPD_HANDLE` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing the requested handle.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.

6.6.31 NPF_IPSecQuery_PolicyHandle : Extract Policy handles based on policy IDs

Syntax

```
NPF_error_t NPF_IPSecQuery_PolicyHandle (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_uint32_t                 countPolicyIDs
      NPF_IN NPF_IPSecPolicy_ID_t         *policyIDArray);
```

Description

Allows extraction of the internal handles associated with a set of user identifiers for that object. The handles are returned via the associated callback using `NPF_IPSecPolicy_Identity_Array_t`.

Note** If the policy handle returned for any element in the array is NPF_IPSEC_POLICY_HANDLE_INVALID, this indicates that the associated policy ID was not recognized as being valid.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `countPolicyIDs`: the number of policy Ids in the ID array
- `policyIDArray`: array of user defined policy identities.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_QUERY_POLICY_HANDLE` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing the requested handle.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.

6.6.32 NPF_IPSecQuery_SAHandle : Extract SA handles based on SA IDs

Syntax

```
NPF_error_t NPF_IPSecQuery_SAHandle (
      NPF_IN NPF_callbackHandle_t         cbHandle,
      NPF_IN NPF_correlator_t             cbCorrelator,
      NPF_IN NPF_errorReporting_t         errorReporting,
      NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
      NPF_IN NPF_uint32_t                 countSAIDs,
      NPF_IN NPF_IPSecSA_ID_t             *arraySAIDs);
```

Description

Allows extraction of the internal handles associated with a set of user identifiers for that object. The handles are returned via the associated callback using `NPF_IPSecSA_Identity_Array_t`.

Note** If the SA handle returned for any element in the array is NPF_IPSEC_SA_HANDLE_INVALID, this indicates that the associated SA ID was not recognized as being valid.

Input Parameters

- `cbHandle`: the registered callback handle.
- `cbCorrelator`: the application's context for this call.
- `errorReporting`: the desired level of feedback.
- `policyHandle`: handle associated with the policy with which the SA is associated.

- `countSAIDs`: the number of SA Ids in the ID array
- `saIDArray`: array of user define SA IDs.

Synchronous Return Codes

As defined in Section 3.6.1, Synchronous Return Codes [Generic Description of IPSec Functions].

Asynchronous response

A callback of type `NPF_IPSEC_QUERY_SA_HANDLE` is generated in response to this function call. A single asynchronous response, `NPF_IPSecAsyncResponse_t`, will be passed to the callback function containing the requested handle.

- NPF_NO_ERROR – The operation was successful.
- NPF_IPSEC_E_NOMEMORY – The system was unable to allocate sufficient memory to complete this operation.
- NPF_IPSEC_E_INVALID_POLICY_HANDLE – The received policy handle was not recognized as being valid.

7 API Capabilities

This section defines the capabilities of the IPSec Service API.

It summarizes the defined APIs and Events and highlights the mandatory (required) and optional features for API compliance.

7.1 API Functions

Function Name	Required?
NPF_IPSecSPD_Create()	Yes
NPF_IPSecSPD_Destroy()	Yes
NPF_IPSecSPD_Bind()	Optional
NPF_IPSecSPD_UnBind()	Optional
NPF_IPSecSPD_Flush()	Optional
NPF_IPSecPolicy_Create()	Yes
NPF_IPSecPolicy_Destroy()	Yes
NPF_IPSecPolicy_BatchCreate()	Optional
NPF_IPSecPolicy_BatchDestroy()	Optional
NPF_IPSecPolicy_Bind()	Optional
NPF_IPSecPolicy_UnBind()	Optional
NPF_IPSecPolicy_BatchBind()	Optional
NPF_IPSecPolicy_BatchUnBind()	Optional
NPF_IPSecSA_Add()	Yes
NPF_IPSecSA_Remove()	Yes
NPF_IPSecSA_ReserveSPI()	Optional
NPF_IPSecSA_ReleaseSPI()	Optional
NPF_IPSecPolicy_GetStats()	Optional
NPF_IPSecSA_GetStats()	Optional
NPF_IPSecRateLimitEvents()	Optional
NPF_IPSecQuery_AllSPDs()	Optional
NPF_IPSecQuery_SPDHandle()	Optional
NPF_IPSecQuery_AllSPDBindings()	Optional
NPF_IPSecQuery_AllPolicies()	Optional
NPF_IPSecQuery_AllPolicyBindings()	Optional
NPF_IPSecQuery_AllSAs()	Optional
NPF_IPSecQuery_PolicyData()	Optional
NPF_IPSecQuery_SAData()	Optional
NPF_IPSecQuery_SPDHandle()	Optional
NPF_IPSecQuery_SAHandle()	Optional
NPF_IPSecQuery_PolicyHandle()	Optional

7.2 API Events

Event Name	Required?
NPF_IPSEC_SA_ACQUIRE	Yes
NPF_IPSEC_SA_EXPIRE_KB_SOFT	Yes
NPF_IPSEC_SA_EXPIRE_KB_HARD	Yes
NPF_IPSEC_SA_EXPIRE_SECS_SOFT	Optional
NPF_IPSEC_SA_EXPIRE_SECS_HARD	Optional
NPF_IPSEC_SA_EXPIRE_PURGED	Yes
NPF_IPSEC_SA_SEQUENCE_OVERFLOW	Optional
NPF_IPSEC_CLEARPACKET_DROPPED	Optional
NPF_IPSEC_REPLAY_PACKET	Optional
NPF_IPSEC_WRONG_SPI	Optional
NPF_IPSEC_AUTH_FAILED	Optional

NPF_IPSEC_DECRYPTION_FAILED	Optional
NPF_IPSEC_INVALID_POLICY	Optional
NPF_IPSEC_POLICY_DISCARD	Optional
NPF_IPSEC_REASSEMBLY_REQD	Optional
NPF_IPSEC_MEM_FULL	Optional

APPENDIX A INFORMATIVE ANNEXES

A.1. HEADER FILE

Header File: npf_ipsec_sapi.h

```
/*
* This header file defines typedefs, constants, and functions
* that apply to the NPF IPSec Service API
*/
#ifndef __NPF_IPSEC_SAPI_H_
#define __NPF_IPSEC_SAPI_H_
#ifdef __cplusplus
extern "C" {
#endif

/*****************************************************************************/
/*
* Misc. Definitions
*/
#define MAX_SIZE_REPLAY_WINDOW 16
#define IPSEC_OPAQUE 0 /* used for ports / protocol to indicate undefined */
#define NPF_IPVERSION4 4 /* IP Version 4 */
#define NPF_IPVERSION6 6 /* IP Version 6 */

/*****************************************************************************/
/*
*     Interface Handle Array
*/
typedef struct
{
      NPF_IN NPF_uint32_t           nifHandles;
      NPF_IN NPF_IfHandle_t         *ifHandleArray;

} NPF_IPSecInterface_Handle_Array_t;

/*
*     The Direction of a given policy / SA
*/
typedef enum
{
      NPF_IPSEC_DIRECTION_INBOUND=1,      /* Inbound policy / SA */
      NPF_IPSEC_DIRECTION_OUTBOUND=2      /* Outbound policy / SA */
} NPF_IPSecDirection_t;

/*
*     SPD ID
*
*     SPD identity selected by the caller
*/
typedef NPF_uint32_t NPF_IPSecSPD_ID_t; /* SPD ID */

/*
*     SPD Handle
*
*     SPD handle selected by the implementation
*/
typedef NPF_uint32_t NPF_IPSecSPD_Handle_t; /* SPD Handle */
/* The following values may not be used to indicate a valid SPD handle */
#define NPF_IPSEC_SPD_HANDLE_INVALID 0

/*
```

```
*     IPSec SPD Handle Array
*
*     SPD handles selected by the implementation
*/
typedef struct
{
      NPF_uint32_t nCount;
      NPF_IPSecSPD_Handle_t* spdHandleArray;
} NPF_IPSecSPD_Handle_Array_t;

/*
*     SPD Identity (ID and Handle)
*
*/
typedef struct
{
      NPF_IPSecSPD_ID_t spdID;
      NPF_IPSecSPD_Handle_t spdHandle;
      NPF_IPSecDirection_t spdDirection;
} NPF_IPSecSPD_Identity_t;

/*
*     SPD Identity Array
*
*/
typedef struct
{
      NPF_uint32_t nCount;
      NPF_IPSecSPD_Identity_t* spdIdentityArray;
} NPF_IPSecSPD_Identity_Array_t;

/*
*     IPSec policy ID
*
*     Policy identity selected by the caller.
*/
typedef NPF_uint32_t NPF_IPSecPolicy_ID_t;

/*
*     IPSec Policy Handle
*
*     Policy handle selected by the implementation
*/
typedef NPF_uint32_t NPF_IPSecPolicy_Handle_t;

/* The following values may not be used to indicate a valid policy handle */
#define NPF_IPSEC_POLICY_HANDLE_INVALID 0

/*
*     IPSec Policy Handle Array
*
*     Policy handles selected by the implementation
*/
typedef struct
{
      NPF_uint32_t nCount;
      NPF_IPSecPolicy_Handle_t * policyHandleArray;
} NPF_IPSecPolicy_Handle_Array_t;

/*
*     Policy Identity (ID and Handle)
*
*/
typedef struct
```

```
{
      NPF_IPSecPolicy_ID_t policyID;
      NPF_IPSecPolicy_Handle_t policyHandle;
} NPF_IPSecPolicy_Identity_t;

/*
*     Policy Identity Array
*
*/
typedef struct
{
      NPF_uint32_t nCount;
      NPF_IPSecPolicy_Identity_t* policyIdentityArray;
} NPF_IPSecPolicy_Identity_Array_t;

/*
*     IPSec SA Identity
*
*     SA identity selected by the caller.
*/
typedef NPF_uint32_t NPF_IPSecSA_ID_t;

/*
*     IPSec SA Handle
*
*     SA handle selected by the implementation
*/
typedef NPF_uint32_t NPF_IPSecSA_Handle_t;

/* The following values may not be used to indicate a valid SA handle */
#define NPF_IPSEC_SA_HANDLE_INVALID 0

/*
*     SA Identity (ID and Handle)
*
*/
typedef struct
{
      NPF_IPSecSA_ID_t policyID;
      NPF_IPSecSA_Handle_t SAHandle;
} NPF_IPSecSA_Identity_t;

/*
*     SA Identity Array
*
*/
typedef struct
{
      NPF_uint32_t nCount;
      NPF_IPSecSA_Identity_t* SAIdentityArray;
} NPF_IPSecSA_Identity_Array_t;

/*
*     SPD Policy Action
*     This is used to indicate the action taken when a given packet
*     matches an SPD rule.
*/
typedef enum
{
      NPF_IPSEC_POLICY_ACTION_IPSEC=1,    /* Apply IPSec */
      NPF_IPSEC_POLICY_ACTION_DISCARD=2,  /* Discard / Drop */
      NPF_IPSEC_POLICY_ACTION_BYPASS=3,   /* Allow to pass */
```

```
} NPF_IPSecPolicy_Action_t;

/*
*     IPSec IP Selector Network Address
*     Note** Basic IP types / structures are defined elsewhere.
*/
typedef union
{
      NPF_IPv4Prefix_t v4;
      NPF_IPv6Prefix_t v6;
} NPF_IPSecPrefix_t;

/*
*     IPSec IP Selector Range for IPv4 and IPv6
*/
typedef struct
{
      NPF_IPv4Address_t start;   /* First address in range */
      NPF_IPv4Address_t end;     /* Last address in range */
} NPF_IPSecIPv4RangeAddr_t;

typedef struct
{
      NPF_IPv6Address_t start;   /* First address in range */
      NPF_IPv6Address_t end;     /* Last address in range */
} NPF_IPSecIPv6RangeAddr_t;

/*
*     Consolidated range union for v4 and v6
*/
typedef union
{
      NPF_IPSecIPv4RangeAddr_t v4;
      NPF_IPSecIPv6RangeAddr_t v6;
} NPF_IPSecRangeAddr_t;

/*
*     IPSec Selector Address type
*/
typedef enum
{
      NPF_IPSEC_ADDR_TYPE_SUBNET=0,
      NPF_IPSEC_ADDR_TYPE_RANGE=1
} NPF_IPSecSelectorAddrType;

/*
*     IPSec Selector Address
*/
typedef struct
{
      NPF_IPSecSelectorAddrType addrType;  /* subnet or range */
      union
      {
            NPF_IPSecPrefix_t       prefixAddr;
            NPF_IPSecRangeAddr_t    rangeAddr;
      } u;
} NPF_IPSecSelectorAddr_t;

/*
```

```
*     IPSec port range
*     Note: A specific port value (a point range) is defined
*     by setting start = end
*/
typedef struct
{
      NPF_uint16_t      start;      /* First port in range */
      NPF_uint16_t      end;        /* Last port in range */
} NPF_IPSecPortRange_t;

/*
*     IPSec Selector
*/
#define NPF_IPSEC_SELECTORRULEFLAG_CLEAR  (0) /* default */
#define NPF_IPSEC_SELECTORRULEFLAG_NOT_SET   (1 << 0)
#define NPF_IPSEC_SELECTORRULEFLAG_AND_SET   (1 << 1)

typedef struct
{
      NPF_uint8_t ruleFlags;              /* rule flag with values above */
      NPF_uint8_t IP_Version;             /* 4 = IPv4, 6 = IPv6 */
      NPF_uint8_t protocol;               /* IP Transp. Protocol or OPAQUE */
      NPF_IPSecSelectorAddr_t srcIP;      /* Src Address */
      NPF_IPSecSelectorAddr_t dstIP;      /* Dst Address */
      NPF_IPSecPortRange_t srcPort;       /* IP Source Port or OPAQUE */
      NPF_IPSecPortRange_t dstPort;       /* IP Destination Port or OPAQUE */
} NPF_IPSecSelector_t;

/*
*     Single IP address
*/
typedef union
{
      NPF_IPv4Address_t v4;               /* IPv4 address */
      NPF_IPv6Address_t v6;               /* IPv6 address */
} NPF_IPSecIPAddress_t;

/*
*     Packet 5-tuple
*/
typedef struct
{
      NPF_uint8_t IP_Version;             /* 4 = IPv4, 6 = IPv6 */
      NPF_uint8_t protocol;               /* IP Transp. Protocol or OPAQUE */
      NPF_IPSecIPAddress_t srcIP;         /* source address of packet */
      NPF_IPSecIPAddress_t dstIP;         /* destination address of packet */
      NPF_uint16_t srcPort;               /* IP Source Port or OPAQUE */
      NPF_uint16_t dstPort;               /* IP Destination Port or OPAQUE */
} NPF_IPSecPacketFields_t;

/*
*     Tunnel Endpoint Addresses
*/
typedef struct
{
      NPF_uint8_t IP_Version;             /* 4 = IPv4, 6 = IPv6 */
      NPF_IPSecIPAddress_t srcIP;
      NPF_IPSecIPAddress_t dstIP;
} NPF_IPSecTunEndAddr_t;
```

```
/*
*     IPSec Error Type
*/
typedef NPF_uint32_t NPF_IPSecErrorType_t;  /* IPSec Error Type */

/*
*     IPSec UDP Encapsulation ports, if used
*/
typedef struct
{
      NPF_uint16_t srcPort;  /* IPSec UDP Src Port */
      NPF_uint16_t dstPort;  /* IPSec UDP Dst Port */
} NPF_IPSecUDP_Ports_t;


/*
*     IPSec SA Parameters
*/
typedef struct
{
    NPF_uint32_t  spi;              /* the SPI / CPI for this SA */
    NPF_uint32_t  flowLabel;        /* Explicit value for IPv6 Flow label */
                                    /* May be used for IPv4 TOS value also */
    NPF_uint32_t  flags;            /* Misc. SA flags i.e. TOS handling */
    NPF_uint8_t   authAlgo;         /* Authentication algorithm */
    NPF_uint8_t   encAlgo;          /* Encryption algorithm */
    NPF_uint32_t  softKbyteLimit;   /* Soft KB expiry */
    NPF_uint32_t  hardKbyteLimit;   /* Hard KB expiry */
    NPF_uint32_t  softSecsLimit;    /* Soft seconds expiry */
    NPF_uint32_t  hardSecsLimit;    /* Hard seconds expiry */
    NPF_IPSecTunEndAddr_t TE_Addr;  /* tunnel endpoint src/dest addr */
    NPF_uint8_t   *authKey;         /* Authentication Key */
    NPF_uint16_t  authKeyLenBits;   /* Algorithm Key Length in bits */
    NPF_uint8_t   *encDecKey;       /* enc / dec key */
    NPF_uint16_t  encDecKeyLenBits; /* Algorithm Key Length in bits */
    NPF_uint8_t   replayWindowSize; /* size of replay window in bytes */
} NPF_IPSecSA_t;

/*
*     IPSec Policy
*/
typedef struct
{
      NPF_IPSecPolicy_ID_t          policyID;         /* Policy ID */
      NPF_IPSecPolicy_Action_t      policyAction;     /* Action */
      NPF_uint32_t                  selectorCount;    /* No of selectors */
      NPF_IPSecSelector_t           *selectorArray;   /* Selector array */
} NPF_IPSecPolicy_t;

/*
*     Rate Limiting Events
*/
typedef enum
{
      NPF_IPSEC_EVENT_LIMIT_TIME=1,       /* Time base limiting */
      NPF_IPSEC_EVENT_LIMIT_COUNT=2       /* Counter base limiting */
} NPF_IPSecEventLimitType_t;

/*
*     IPSec Event
*/
```

```
typedef enum
{
      NPF_IPSEC_SA_ACQUIRE = 1,           /* SPD rule found but no SA */
      NPF_IPSEC_SA_EXPIRE_KB_SOFT = 2,    /* SA KB soft Expired */
      NPF_IPSEC_SA_EXPIRE_KB_HARD = 3,    /* SA KB hard Expired */
      NPF_IPSEC_SA_EXPIRE_SECS_SOFT = 4,  /* SA seconds soft Expired */
      NPF_IPSEC_SA_EXPIRE_SECS_HARD = 5,  /* SA seconds hard Expired */
      NPF_IPSEC_SA_PURGED = 6,            /* SA forcibly removed */
      NPF_IPSEC_SA_SEQUENCE_OVERFLOW = 7, /* Sequence # overflow */
      NPF_IPSEC_CLEARPACKET_DROPPED = 8,  /* Clear text Packet dropped */
      NPF_IPSEC_REPLAY_PACKET = 9,        /* Replay packet caught */
      NPF_IPSEC_WRONG_SPI = 10,           /* Unknown SPI-value in packet */
      NPF_IPSEC_AUTH_FAILED = 11,         /* Authentication failed */
      NPF_IPSEC_DECRYPTION_FAILED = 12,   /* IPSec decryption failed */
      NPF_IPSEC_INVALID_POLICY = 13,      /* Decryption-policy mismatch */
      NPF_IPSEC_POLICY_DISCARD = 14,      /* Discard policy */
      NPF_IPSEC_REASSEMBLY_REQD = 15,     /* Fragmented packet received */
      NPF_IPSEC_MEM_FULL = 16             /* A crypto NPU ran out of memory*/
} NPF_IPSecEvent_t;

#define IPSEC_MAX_EVENT_TYPES 16

/*
*     IPSec event bitmask used in the event registration call.
*/
typedef NPF_uint32_t NPF_IPSecEventMask_t;

/*
* The following values can be set for the IPSecEventMask
*/

#define NPF_IPSEC_EVENT_ALL_DISABLE                   (0) /* disable all */
#define NPF_IPSEC_EVENT_SA_ACQUIRE_ENABLE             (1 << 0)
#define NPF_IPSEC_EVENT_SA_EXPIRE_KB_SOFT_ENABLE      (1 << 1)
#define NPF_IPSEC_EVENT_SA_EXPIRE_KB_HARD_ENABLE      (1 << 2)
#define NPF_IPSEC_EVENT_SA_EXPIRE_SECS_SOFT_ENABLE    (1 << 3)
#define NPF_IPSEC_EVENT_SA_EXPIRE_SECS_HARD_ENABLE    (1 << 4)
#define NPF_IPSEC_EVENT_SA_PURGED_ENABLE              (1 << 5)
#define NPF_IPSEC_EVENT_SA_SEQUENCE_OVERFLOW_ENABLE   (1 << 6)
#define NPF_IPSEC_EVENT_CLEARPACKET_DROPPED_ENABLE    (1 << 7)
#define NPF_IPSEC_EVENT_REPLAY_PACKET_ENABLE          (1 << 8)
#define NPF_IPSEC_EVENT_WRONG_SPI_ENABLE              (1 << 9)
#define NPF_IPSEC_EVENT_AUTH_FAILED_ENABLE            (1 << 10)
#define NPF_IPSEC_EVENT_DECRYPTION_FAILED_ENABLE      (1 << 11)
#define NPF_IPSEC_EVENT_INVALID_POLICY_ENABLE         (1 << 12)
#define NPF_IPSEC_EVENT_POLICY_DISCARD_ENABLE         (1 << 13)
#define NPF_IPSEC_EVENT_REASSEMBLY_REQD_ENABLE        (1 << 14)
#define NPF_IPSEC_EVENT_MEM_FULL_ENABLE               (1 << 15)

#define NPF_IPSEC_EVENT_ALL_ENABLE                    0xFFFFFFFF

/*
*     IPSec call handle
*/
typedef NPF_uint32_t NPF_IPSecEventCallHandle_t; /* Event call handle */

typedef struct
{
      NPF_IPSecEvent_t              eventid;          /* Event ID */
      NPF_IPSecEventLimitType_t     limitType;        /* Limit type */
```

```
      union
      {
            NPF_uint32_t numPerSec; /* Event frequency in time */
            NPF_uint32_t nCount;    /* Generate 1 event for every
                                       nCount encounters */
      }u;
} NPF_IPSecEventLimit_t;

/*
*     IPSec SPD Policy statistics
*/
typedef struct
{
    NPF_IPSecSPD_ID_t     spdID;    /* Identity of SPD containing policy */
    NPF_IPSecPolicy_ID_t  policyID; /* Identity of policy */
    NPF_uint32_t  policyErrors;     /* policy validation (selector) errors */
    NPF_uint32_t  acquiredSAs;      /* total number of SAs acquired */
    NPF_uint32_t  exceptions;       /* total exceptions generated for SA */

} NPF_IPSecPolicy_Stats_t;

/*
*     IPSec SA statistics
*/
typedef struct
{
    NPF_uint32_t  HMAC_Errors;       /* HMAC (hash) errors -inbound only- */
    NPF_uint32_t  decryptErrors;     /* decryption  errors -inbound only- */
    NPF_uint32_t  replayErrors;      /* replay attacks -inbound only- */
    NPF_uint32_t  selectorErrors;    /* SA selectors validation errors */
    NPF_uint64_t  packetCount;       /* packet count */
    /* The above counters may be reset by a reset_SA function call */
    NPF_uint64_t  bytesUsed;         /* # bytes the SA has been applied to */
    NPF_uint64_t  bytesRemaining;    /* # bytes remaining the SA may be
                                        applied to */
    NPF_uint32_t  secsUsed;          /* # secs used for this SA */
    NPF_uint32_t  secsRemaining;     /* # secs remaining before SA expires */
    NPF_uint64_t  sequenceNo;        /* sequence number (Tx or Rx) */
    NPF_uint32_t  exceptions;        /* total exceptions generated for SA */
    /*  copy of replay bitmap -inbound only- */
    NPF_uint32_t  replayBitmap[MAX_SIZE_REPLAY_WINDOW];

} NPF_IPSecSA_Stats_t;

/*
*     IPSec SA Bundle Stats
*/
typedef struct
{
      NPF_IPSecSA_ID_t saID;        /* User identifier for this SA */
      NPF_uint32_t saCount;         /* The number of SAs in the bundle */
      NPF_IPSecSA_Stats_t* saBundleStats; /* pointer to the SA bundle data */
} NPF_IPSecSA_BundleStats_t;

/*
* completion callback types
*/
typedef enum
{
      NPF_IPSEC_SPD_CREATE    = 1,
      NPF_IPSEC_SPD_DESTROY,
      NPF_IPSEC_SPD_BIND,
```

```
      NPF_IPSEC_SPD_UNBIND,
      NPF_IPSEC_SPD_FLUSH,
      NPF_IPSEC_POLICY_CREATE,
      NPF_IPSEC_POLICY_DESTROY,
      NPF_IPSEC_POLICY_BATCH_CREATE,
      NPF_IPSEC_POLICY_BATCH_DESTROY,
      NPF_IPSEC_POLICY_BIND,
      NPF_IPSEC_POLICY_UNBIND,
      NPF_IPSEC_POLICY_BATCH_BIND,
      NPF_IPSEC_POLICY_BATCH_UNBIND,
      NPF_IPSEC_POLICY_CHANGEPRIORITY,
      NPF_IPSEC_SA_ADD,
      NPF_IPSEC_SA_REMOVE,
      NPF_IPSEC_SA_RESERVESPI,
      NPF_IPSEC_SA_RELEASESPI,
      NPF_IPSEC_RATELIMIT_EVENTS,
      NPF_IPSEC_POLICY_GET_STATS,
      NPF_IPSEC_SA_GET_STATS,
      NPF_IPSEC_QUERY_ALL_SPDS,
      NPF_IPSEC_QUERY_ALL_SPD_BINDINGS,
      NPF_IPSEC_QUERY_ALL_POLICIES,
      NPF_IPSEC_QUERY_ALL_POLICY_BINDINGS,
      NPF_IPSEC_QUERY_ALL_SAS,
      NPF_IPSEC_QUERY_POLICY_DATA,
      NPF_IPSEC_QUERY_SA_DATA,
      NPF_IPSEC_QUERY_SPD_HANDLE,
      NPF_IPSEC_QUERY_POLICY_HANDLE,
      NPF_IPSEC_QUERY_SA_HANDLE
} NPF_IPSecCallbackType_t;

#define IPSEC_MAX_CALLBACK_TYPES 31

/*
* An asynchronous response contains an SPD handle,
* an error or success code, a direction indicator,
* and in most cases a function-specific type embedded
* in a union. One or more of these responses
* is passed to the callback function as an array
* within the NPF_IPSecCallbackData_t structure (below).
*/
typedef struct /* Asynchronous Response Structure */
{
      NPF_IPSecErrorType_t error;         /* Error code for this response */
      NPF_IPSecDirection_t direction;
      union /* Function-specific structures: */
      {
            /*
            *     The SPD Handle is returned for the following callbacks
            *
            *     NPF_IPSEC_SPD_CREATE
            *     NPF_IPSEC_SPD_BIND
            *     NPF_IPSEC_SPD_UNBIND
            *     NPF_IPSEC_SPD_FLUSH
            *     NPF_IPSEC_SPD_DESTROY
            *
            *     and optionally for the following callbacks, if the error
            *     code pertains to an invalid SPD handle
            *
            *     NPF_IPSEC_POLICY_CREATE
            *     NPF_IPSEC_POLICY_BIND
            *     NPF_IPSEC_POLICY_UNBIND
```

```
*      NPF_IPSEC_POLICY_DESTROY
*
*      NPF_IPSEC_POLICY_BATCH_CREATE
*      NPF_IPSEC_POLICY_BATCH_DESTROY
*      NPF_IPSEC_POLICY_BATCH_BIND
*      NPF_IPSEC_POLICY_BATCH_UNBIND
*
*      NPF_IPSEC_SA_ADD
*      NPF_IPSEC_SA_REMOVE
*
*      And for the following error codes when unbinding
*      NPF_IPSEC_E_UNBIND_FAILED_LASTBINDING
*      NPF_IPSEC_E_UNBIND_FAILED_BOUNDTOALL
*/
NPF_IPSecSPD_Handle_t spdHandle;

/*
*      The policy handle is returned for the following callbacks
*      NPF_IPSEC_POLICY_CREATE
*      NPF_IPSEC_POLICY_BIND
*      NPF_IPSEC_POLICY_UNBIND
*      NPF_IPSEC_POLICY_CHANGEPRIORITY
*      NPF_IPSEC_POLICY_DESTROY
*
*      NPF_IPSEC_POLICY_BATCH_DESTROY
*      NPF_IPSEC_POLICY_BATCH_BIND
*      NPF_IPSEC_POLICY_BATCH_UNBIND
*
*      and also for the following callbacks
*      NPF_IPSEC_SA_ADD or
*      NPF_IPSEC_SA_REMOVE,
*      NPF_IPSEC_SA_RESERVESPI,
*      NPF_IPSEC_SA_RELEASESPI, if the return code was
*      NPF_IPSEC_E_INVALID_POLICY_HANDLE or
*      NPF_IPSEC_E_DUPLICATE_POLICY_HANDLE
*      NPF_IPSEC_E_UNBIND_FAILED_LASTBINDING
*      NPF_IPSEC_E_UNBIND_FAILED_BOUNDTOALL
*/
NPF_IPSecPolicy_Handle_t policyHandle;

/*
*      The IPSec Selector is returned for a callback of
*      NPF_IPSEC_POLICY_CREATE
*      NPF_IPSEC_POLICY_BATCH_CREATE
*      with an error of
*      NPF_IPSEC_E_INVALID_SELECTOR. This structure will
*      contain a single selector, which was deemed invalid
*/
NPF_IN NPF_IPSecSelector_t    selector;

/*
*      The SA structure is returned in the NPF_IPSEC_SA_ADD
*      in the case the SA could not be added because
*      of some errors in the SA definition
*      or in the case of NPF_IPSEC_QUERY_SA_DATA
*/
NPF_IPSecSA_t SA;

/*
```

```
*     The SA Handle is returned when an SA is added,
*     NPF_IPSEC_SA_ADD,
*     when the SA is removed, NPF_IPSEC_SA_REMOVE
*/
NPF_IPSecSA_Handle_t    saHandle;

/*
*     The SPD policy counters structure is returned in the
*     NPF_IPSEC_POLICY_GET_STATISTICS call
*/
NPF_IPSecPolicy_Stats_t policyStats;

/*
*     The SA counters structure is returned in the
*     NPF_IPSEC_SA_GET_STATISTICS call
*/
NPF_IPSecSA_BundleStats_t saStats;

/*
* The SPI is returned in the NPF_IPSEC_RESERVE_SPI call
*/
NPF_uint32_t spi;

/* In the case of NPF_IPSEC_RATELIMIT_EVENTS, there is no
* additional information provided via this union
*/

/* A interface handle may be returned in the case
* the interface handle provided in the API was invalid
* NPF_IPSEC_E_INVALID_IF_HANDLE
*/
NPF_IfHandle_t ifHandle;

/*
* SPD ID may be returned for any function
* if the returned error is
* NPF_IPSEC_E_INVALID_SPD_ID
* NPF_IPSEC_E_SPD_ID_ALREADY_REGISTERED
*/
NPF_IPSecSPD_ID_t spdID;

/*
* Policy ID may be returned for any function
* if the returned error is
* NPF_IPSEC_E_INVALID_POLICY_ID
* NPF_IPSEC_E_POLICY_ID_ALREADY_REGISTERED
* NPF_IPSEC_E_DUPLICATE_POLICY_ID
*/
NPF_IPSecPolicy_ID_t policyID;

/*
* SA ID may be returned for any function
* if the returned error is
* NPF_IPSEC_E_INVALID_SA_ID
* NPF_IPSEC_E_SA_ID_ALREADY_REGISTERED
*/
NPF_IPSecSA_ID_t saID;
```

```
            /*
            * An array of policy handles is returned for the batch function
            * NPF_IPSEC_POLICY_BATCH_CREATE
            */
            NPF_IPSecPolicy_Handle_Array_t policyHandles;

            /*
            * NPF_IPSEC_QUERY_ALL_SPDS
            * NPF_IPSEC_QUERY_ALL_POLICY_BINDINGS
            * NPF_IPSEC_QUERY_SPD_HANDLE
            */
            NPF_IPSecSPD_Identity_Array_t spdDataArray;

            /*
            * NPF_IPSEC_QUERY_ALL_SPD_BINDINGS
            */
            NPF_IPSecInterface_Handle_Array_t interfaceDataArray;

            /*
            * NPF_IPSEC_QUERY_ALL_POLICIES
            * NPF_IPSEC_QUERY_POLICY_HANDLE
            */
            NPF_IPSecPolicy_Identity_Array_t policyDataArray;

            /*
            * NPF_IPSEC_QUERY_ALL_SAS
            * NPF_IPSEC_QUERY_SA_HANDLE
            */
            NPF_IPSecSA_Identity_Array_t saDataArray;

            /*
            * NPF_IPSEC_QUERY_POLICY_DATA
            */
            NPF_IPSecPolicy_t policyData;

      } u;
} NPF_IPSecAsyncResponse_t;

typedef struct
{
      NPF_IPSecCallbackType_t type;       /* Response to which function */
      NPF_boolean_t           allOK;      /* TRUE is all completed OK */
      NPF_uint32_t            nResp;      /* Number of responses in array */
      NPF_IPSecAsyncResponse_t *resp;     /* Pointer to response structures*/
} NPF_IPSecCallbackData_t;

/*
*      Information about missing SA
*/
typedef struct {
      NPF_IPSecSPD_ID_t spdID;              /* client side SPD ID */
      NPF_IPSecPolicy_ID_t policyID;        /* client side policy ID */
      NPF_IPSecPacketFields_t packet5Tuple; /* Fields that match selector */
      NPF_uint32_t acquireContext;          /* Acquire Context */
} NPF_IPSecSA_AcquireInfo_t;

/*
*     SA Expired information used by NPF_IPSEC_SA_EXPIRE_KB_SOFT,
*     NPF_IPSEC_SA_EXPIRE_KB_HARD, NPF_IPSEC_SA_EXPIRE_SECS_SOFT,
*     NPF_IPSEC_SA_EXPIRE_SECS_HARD or NPF_IPSEC_SA_PURGED
```

```
*/
typedef struct {
      NPF_IPSecSA_ID_t saID;              /* client side ID for SA */
} NPF_IPSecSA_ExpireInfo_t;

/*
*     Information about the dropped packet, should be used every time a
*     packet is dropped i.e. it shall be used together with
*     NPF_IPSEC_CLEARPACKET_DROPPED and
*     NPF_IPSEC_INVALID_POLICY
*     Depending on the event, one or more IDs must be provided. SPDid and
*     PolicyID must be provided for NPF_IPSEC_CLEARPACKET_DROPPED and
*     NPF_IPSEC_INVALID_POLICY; SAId must be provided for all events except
*     NPF_IPSEC_CLEARPACKET_DROPPED. If no value is required, any value will
*     serve as dummy value.
*/
typedef struct {
      NPF_IPSecSPD_ID_t spdID;                  /* client side SPD ID */
      NPF_IPSecPolicy_ID_t policyID;            /* client side policy id */
      NPF_IPSecSA_ID_t saID;                    /* SA handle */
      NPF_IPSecPacketFields_t packet5Tuple;     /*offending packet 5-tuple */
} NPF_IPSecPacketDropped_t;

/*
*     Information about crypto packet dropped for events:
*     NPF_IPSEC_AUTH_FAILED
*     NPF_IPSEC_DECRYPTION_FAILED
*     NPF_IPSEC_SA_SEQUENCE_OVERFLOW
*     NPF_IPSEC_POLICY_DISCARD
*     NPF_IPSEC_REASSEMBLY_REQD
*/
typedef struct {
      NPF_uint8_t IP_Version;             /* IPv4 = 4, IPv6 = 6 */
      NPF_IPSecIPAddress_t srcIP;         /* Src Address */
      NPF_IPSecIPAddress_t dstIP;         /* Dst Address */
      NPF_uint8_t protocol;               /* ESP = 50, AH = 51, COMP = 108 */
      NPF_uint32_t SPI;                   /* SPI value */
      NPF_uint32_t flowLabel;             /* IPv6 Flow label, 0 for IPv4 */
      NPF_uint8_t seqNo;                  /* Anti replay counter */
      NPF_uint8_t count;                  /* Repetition count */
} NPF_IPSecCryptoDropped_t;

/*
*     IPSec Event Data
*/
typedef struct
{
      NPF_IPSecEvent_t eventType;
      union
      {

            NPF_IPSecSA_AcquireInfo_t     acquire;
            NPF_IPSecSA_ExpireInfo_t      expire;
            NPF_IPSecPacketDropped_t      packet;
            NPF_IPSecCryptoDropped_t      crypto;
      } event;
} NPF_IPSecEventData_t;

/*
*     IPSec Event Array
```

```
*/

typedef struct
{
      NPF_uint16_t            n_data;     /* Number of events in array */
      NPF_IPSecEventData_t    *eventDataArray;  /* Array of event
                                                   notifications */
} NPF_IPSecEventArray_t;

/*********************************************************************/
/*
*     Asynchronous error codes (returned in function callbacks)
*/

/*
*     IPSec reserved error codes in relation to other NPF APIs
*     Note** The maximum range is 100
*/
#define NPF_IPSEC_BASE_ERR 600 /* Base value of 100 wrt other NPF codes */

/*
*     These are generic error codes, that can be returned in any callback
*/
#define NPF_IPSEC_GENERIC_ERROR_CODE_COUNT 20 /* Should be enough */

/* The Interface handle provided was not recognized as being valid */
#define NPF_IPSEC_E_INVALID_IF_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + 1)

/* Already registered */
#define NPF_IPSEC_E_ALREADY_REGISTERED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + 2)

/* Optional feature not supported */
#define NPF_IPSEC_E_OPTIONAL_FEATURE_NOT_SUPPORTED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + 3)

/* System unable to allocate sufficient memory to complete this operation */
#define NPF_IPSEC_E_NOMEMORY  ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + 4)

/*
* Invalid IP Address error codes are defined in the IF Management API
* i.e.
*
* Invalid IP address : NPF_IF_E_INVALID_IPADDR
* Invalid IP net prefix length : NPF_IF_E_INVALID_NETPLEN
*/

/* Attempt to bind more than one SPD to an interface */
#define NPF_IPSEC_E_IF_ALREADY_BOUND \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 1)


/* Invalid SPD handle */
#define NPF_IPSEC_E_INVALID_SPD_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 2)
```

```
/* Invalid policy handle */
#define NPF_IPSEC_E_INVALID_POLICY_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 3)

/* Invalid SA handle */
#define NPF_IPSEC_E_INVALID_SA_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 4)

/* Invalid SPD ID - used in query functions */
#define NPF_IPSEC_E_INVALID_SPD_ID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 5)

/* Invalid policy ID - used in query functions */
#define NPF_IPSEC_E_INVALID_POLICY_ID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 6)

/* Invalid SA ID  - used in query functions */
#define NPF_IPSEC_E_INVALID_SA_ID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 7)

/* Invalid Policy Priority */
#define NPF_IPSEC_E_INVALID_POLICY_PRIORITY \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 8)

/* Invalid Policy Action */
#define NPF_IPSEC_E_INVALID_POLICY_ACTION \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 9)

/* Invalid selector, when attempting to add a policy */
#define NPF_IPSEC_E_INVALID_SELECTOR \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
      NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 10)

/* Invalid encryption algorithm, when attempting to add an SA */
#define NPF_IPSEC_E_INVALID_ENC_ALGO \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 11)

/* Invalid authentication algorithm, when attempting to add an SA */
#define NPF_IPSEC_E_INVALID_AUTH_ALGO \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 12)

/* Invalid compression algorithm, when attempting to add an SA */
#define NPF_IPSEC_E_INVALID_COMPRESS_ALGO \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 13)

/* NULL auth. and NULL encr. algorithms, when attempting to add an SA */
#define NPF_IPSEC_E_NULL_AUTH_NULL_ENC_ALGO \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 14)
```

```
/* Invalid encryption algorithm keylen, when attempting to add an SA */
#define NPF_IPSEC_E_INVALID_ENC_ALGO_KEYLEN \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 15)

/* Invalid authentication algorithm keylen, when attempting to add an SA */
#define NPF_IPSEC_E_INVALID_AUTH_ALGO_KEYLEN \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 16)

/* Cannot enable anti-replay when no authentication algorithm defined */
#define NPF_IPSEC_E_NO_REPLAY \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 17)

/* The SA SPI value provided is unacceptable to the implementation */
#define NPF_IPSEC_E_BAD_SPI \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 18)

/* ReserveSPI mode of operation not available */
#define NPF_IPSEC_E_RESERVESPIMODE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 19)

#define NPF_IPSEC_E_SPIINUSE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 20)

/* Duplicate IF handle specified */
#define NPF_IPSEC_E_DUPLICATE_IF_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 21)

/* Duplicate SPD handle specified */
#define NPF_IPSEC_E_DUPLICATE_SPD_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 22)

/* Duplicate policy handle specified */
#define NPF_IPSEC_E_DUPLICATE_POLICY_HANDLE \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 23)

/* Duplicate event ID specified */
#define NPF_IPSEC_E_DUPLICATE_EVENTID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 24)

/* Unrecognized event ID specified */
#define NPF_IPSEC_E_BAD_EVENTID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 25)

/* Failed to unbind an object */
#define NPF_IPSEC_E_UNBIND_FAILED_BOUNDTOALL \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 26)
```

```
/* Failed to unbind an object */
#define NPF_IPSEC_E_UNBIND_FAILED_LASTBINDING \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 27)

/* Failed to bind an object */
#define NPF_IPSEC_E_BIND_FAILED_BOUNDTOALL \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 28)

/* Failed to bind an object */
#define NPF_IPSEC_E_BIND_FAILED_ALREADYBOUND \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 29)

/* SPD ID already registered */
#define NPF_IPSEC_E_SPD_ID_ALREADY_REGISTERED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 30)

/* Policy ID already registered */
#define NPF_IPSEC_E_POLICY_ID_ALREADY_REGISTERED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 31)

/* SA ID already registered */
#define NPF_IPSEC_E_SA_ID_ALREADY_REGISTERED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 32)

/* Duplicate policy ID */
#define NPF_IPSEC_E_DUPLICATE_POLICY_ID \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
       NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 33)

/* Duplicate selector */
#define NPF_IPSEC_E_DUPLICATE_SELECTOR \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
      NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 34)

/* Selector rule flag unsupported */
#define NPF_IPSEC_E_SELECTOR_RULEFLAG_UNSUPPORTED \
      ((NPF_IPSecErrorType_t) NPF_IPSEC_BASE_ERR + \
      NPF_IPSEC_GENERIC_ERROR_CODE_COUNT + 35)

/*
* Core definitions
*/

/* IPSec Protocol */
#define NPF_IPSEC_PROTOCOL_ESP      0x01   /* RFC-2406 */
#define NPF_IPSEC_PROTOCOL_AH       0x02   /* RFC-2402 */
#define NPF_IPSEC_PROTOCOL_IPCOMP   0x04   /* IPCOMP */
#define NPF_IPSEC_PROTOCOL_V2_BIS   0x08   /* ESP / AH bis v2 */

/* DF Bit Handling */
#define NPF_IPSEC_DF_COPY           0x0   /* copy */
#define NPF_IPSEC_DF_CLEAR          0x1   /* clear */
```

```
#define NPF_IPSEC_DF_SET            0x2   /* set */

/* IPv4 TOS handling */
#define NPF_IPSEC_QOS_TOS_COPY      0x0   /* copy */
#define NPF_IPSEC_QOS_TOS_CLEAR     0x1   /* clear */
#define NPF_IPSEC_QOS_TOS_SET       0x2   /* set */

/* IPv6 DSCP / FLowlabel handling */
#define NPF_IPSEC_QOS_DSCP_COPY           0x00  /* copy */
#define NPF_IPSEC_QOS_DSCP_CLEAR          0x01  /* clear */
#define NPF_IPSEC_QOS_DSCP_SET            0x02  /* set */

#define NPF_IPSEC_QOS_FLOWLABEL_COPY      0x00  /* copy */
#define NPF_IPSEC_QOS_FLOWLABEL_CLEAR     0x10  /* clear */
#define NPF_IPSEC_QOS_FLOWLABEL_SET       0x20  /* set */

/* Tunnel / Transport Mode*/
#define NPF_IPSEC_SA_SAFLAGS_TUNNELMODE         0x0
#define NPF_IPSEC_SA_SAFLAGS_TRANSPORTMODE      0x1

/* Replay protection on / off */
#define NPF_IPSEC_SA_SAFLAGS_REPLAY_ON    0x0
#define NPF_IPSEC_SA_SAFLAGS_REPLAY_OFF   0x1

/* control lifetime in seconds */
#define NPF_IPSEC_SA_SAFLAGS_LIFESECS_ON  0x0
#define NPF_IPSEC_SA_SAFLAGS_LIFESECS_OFF 0x1

/* algorithms */
#define NPF_IPSEC_AALG_NONE           0
#define NPF_IPSEC_AALG_MD5HMAC        2
#define NPF_IPSEC_AALG_SHA1HMAC       3
#define NPF_IPSEC_AALG_AESXCBC        4

#define NPF_IPSEC_AALG_NONE_KEYBITS_LENGTH        0
#define NPF_IPSEC_AALG_MD5HMAC_KEYBITS_LENGTH     128
#define NPF_IPSEC_AALG_SHA1HMAC_KEYBITS_LENGTH    160
#define NPF_IPSEC_AALG_AESXCBC_KEYBITS_LENGTH     128

#define NPF_IPSEC_EALG_NONE           0
#define NPF_IPSEC_EALG_DESCBC         2
#define NPF_IPSEC_EALG_3DESCBC        3
#define NPF_IPSEC_EALG_NULL           11
#define NPF_IPSEC_EALG_AES            12

#define NPF_IPSEC_EALG_DESCBC_KEYBITS_LENGTH      64
#define NPF_IPSEC_EALG_3DESCBC_KEYBITS_LENGTH     192
#define NPF_IPSEC_EALG_NULL_KEYBITS_LENGTH        0
#define NPF_IPSEC_EALG_AES128_KEYBITS_LENGTH      128
#define NPF_IPSEC_EALG_AES192_KEYBITS_LENGTH      192
#define NPF_IPSEC_EALG_AES256_KEYBITS_LENGTH      256

/*****************************************************************************/
/************************ Function Prototypes ********************************/
/*****************************************************************************/


/*****************************************************************************/
/* Completion Callback */
```

```
/*****************************************************************************/

/*
* Callback Signature
*/

typedef void (*NPF_IPSecCallbackFunc_t)(
                  NPF_IN NPF_userContext_t            userContext,
                  NPF_IN NPF_correlator_t             correlator,
                  NPF_IN NPF_IPSecCallbackData_t      callbackData);

/*
* Callback Registration
*/

NPF_error_t NPF_IPSecRegister(NPF_IN  NPF_userContext_t     userContext,
                              NPF_IN  NPF_IPSecCallbackFunc_t     cbFunc,
                              NPF_OUT NPF_callbackHandle_t*       cbHandle);

/*
* Callback De-registration
*/

NPF_error_t NPF_IPSecDeregister(NPF_IN NPF_callbackHandle_t cbHandle);

/*****************************************************************************/
/* Event Handling */
/*****************************************************************************/

/*
* Event Handling function signature
*/

typedef void (*NPF_IPSecEventCallFunc_t)(
                              NPF_IN NPF_userContext_t      userContext,
                              NPF_IN NPF_IPSecEventArray_t  eventArray);

/*
* Registration
*/

NPF_error_t NPF_IPSecEventRegister(
            NPF_IN      NPF_userContext_t             userContext,
            NPF_IN      NPF_IPSecEventCallFunc_t      eventCallFunc,
            NPF_IN      NPF_IPSecEventMask_t          eventMask,
            NPF_OUT     NPF_IPSecEventCallHandle_t    *eventCallHandle);

/*
* De-registration
*/

NPF_error_t NPF_IPSecEventDeregister(
            NPF_IN  NPF_IPSecEventCallHandle_t  eventCallHandle);

/*
* Rate-limit Events
*/

NPF_error_t NPF_IPSecRateLimitEvents (
```

```
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_uint32_t                 countEventData,
            NPF_IN NPF_IPSecEventLimit_t        *eventLimitArray);

/*************************************************************************/
/* SPD Handling */
/*************************************************************************/

/*
* SPD Create
*/

NPF_error_t NPF_IPSecSPD_Create (
            NPF_IN NPF_callbackHandle_t   cbHandle,
            NPF_IN NPF_correlator_t       cbCorrelator,
            NPF_IN NPF_errorReporting_t   errorReporting,
            NPF_IN NPF_IPSecDirection_t   direction,
            NPF_IN NPF_uint32_t           nifHandles,
            NPF_IN NPF_IfHandle_t         *ifHandleArray,
            NPF_IN NPF_IPSecSPD_ID_t      spdID);

/*
* SPD Destroy
*/

NPF_error_t NPF_IPSecSPD_Destroy (
            NPF_IN NPF_callbackHandle_t   cbHandle,
            NPF_IN NPF_correlator_t       cbCorrelator,
            NPF_IN NPF_errorReporting_t   errorReporting,
            NPF_IN NPF_uint32_t           nSPDHandles,
            NPF_IN NPF_IPSecSPD_Handle_t  *spdHandleArray);

/*
* SPD Bind
*/

NPF_error_t NPF_IPSecSPD_Bind (
            NPF_IN NPF_callbackHandle_t   cbHandle,
            NPF_IN NPF_correlator_t       cbCorrelator,
            NPF_IN NPF_errorReporting_t   errorReporting,
            NPF_IN NPF_IPSecSPD_Handle_t  spdHandle,
            NPF_IN NPF_uint32_t           nifHandles,
            NPF_IN NPF_IfHandle_t         *ifHandleArray);

/*
* SPD UnBind
*/

NPF_error_t NPF_IPSecSPD_UnBind (
            NPF_IN NPF_callbackHandle_t   cbHandle,
            NPF_IN NPF_correlator_t       cbCorrelator,
            NPF_IN NPF_errorReporting_t   errorReporting,
            NPF_IN NPF_IPSecSPD_Handle_t  spdHandle,
            NPF_IN NPF_uint32_t           nifHandles,
            NPF_IN NPF_IfHandle_t         *ifHandleArray);

/*
```

```
* SPD Flush
*/

NPF_error_t NPF_IPSecSPD_Flush (
            NPF_IN NPF_callbackHandle_t   cbHandle,
            NPF_IN NPF_correlator_t       cbCorrelator,
            NPF_IN NPF_errorReporting_t   errorReporting,
            NPF_IN NPF_uint32_t           nSPDHandles,
            NPF_IN NPF_IPSecSPD_Handle_t  *spdHandleArray);

/*************************************************************************/
/* Policy Handling */
/*************************************************************************/

/*
* Policy Create
*/

NPF_error_t NPF_IPSecPolicy_Create (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_t            *policy,
            NPF_IN NPF_uint32_t                 nHandles,
            NPF_IN NPF_IPSecSPD_Handle_t        *spdHandleArray,
            NPF_IN NPF_IPSecDirection_t         direction,
            NPF_IN NPF_uint8_t                  *prioritiesArray);

/*
* Policy Destroy
*/

NPF_error_t NPF_IPSecPolicy_Destroy (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle);

/*
* Batch Policy Create
*/

NPF_error_t NPF_IPSecPolicy_BatchCreate (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_uint32_t                 nPolicies,
            NPF_IN NPF_IPSecPolicy_t            *policyArray,
            NPF_IN NPF_IPSecSPD_Handle_t        spdHandle,
            NPF_IN NPF_IPSecDirection_t         direction,
            NPF_IN NPF_uint32_t                 *prioritiesArray);

/*
* Batch Policy Destroy
*/

NPF_error_t NPF_IPSecPolicy_BatchDestroy (
            NPF_IN NPF_callbackHandle_t         cbHandle,
```

```
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_uint32_t                 nPolicies,
            NPF_IN NPF_IPSecPolicy_Handle_t     *policyHandleArray);

/*
* Policy Bind
*/

NPF_error_t NPF_IPSecPolicy_Bind (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
            NPF_IN NPF_uint32_t                 nHandles,
            NPF_IN NPF_IPSecSPD_Handle_t        *spdHandleArray,
            NPF_IN NPF_uint8_t                  *prioritiesArray);


/*
* Policy UnBind
*/

NPF_error_t NPF_IPSecPolicy_UnBind (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
            NPF_IN NPF_uint32_t                 nSPDHandles,
            NPF_IN NPF_IPSecSPD_Handle_t        *spdHandleArray);

/*
* Batch Policy Bind
*/

NPF_error_t NPF_IPSecPolicy_BatchBind (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_uint32_t                 nPolicies,
            NPF_IN NPF_IPSecPolicy_Handle_t     *policyHandleArray,
            NPF_IN NPF_IPSecSPD_Handle_t        spdHandle,
            NPF_IN NPF_IPSecDirection_t         direction,
            NPF_IN NPF_uint32_t                 *prioritiesArray);

/*
* Batch Policy UnBind
*/

NPF_error_t NPF_IPSecPolicy_BatchUnBind (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_uint32_t                 nPolicies,
            NPF_IN NPF_IPSecPolicy_Handle_t     *policyHandleArray,
            NPF_IN NPF_IPSecSPD_Handle_t        spdHandle);

/*
* Policy Change Priority
```

```
*/

NPF_error_t NPF_IPSecPolicy_ChangePriority (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecSPD_Handle_t        spdHandle,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
            NPF_IN NPF_uint8_t                  newPriority);

/*************************************************************************/
/* SA Handling */
/*************************************************************************/

/*
* SA Add
*/

NPF_error_t NPF_IPSecSA_Add (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
            NPF_IN NPF_IPSecSA_ID_t             saID,
            NPF_IN NPF_uint32_t                 nSA_Count,
            NPF_IN NPF_IPSecSA_t                *saArray,
            NPF_IN NPF_IPSecUDP_Ports_t         udpPorts,
            NPF_IN NPF_uint32_t                 selectorCount,
            NPF_IN NPF_IPSecSelector_t          *selectorArray,
            NPF_IN NPF_IPSecSA_Handle_t         previousSA,
            NPF_IN NPF_IPSecDirection_t         direction,
            NPF_IN NPF_uint32_t                 acquireContext,
            NPF_IN NPF_uint32_t                 vendorSpecHint);

/*
* SA Remove
*/

NPF_error_t NPF_IPSecSA_Remove (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
            NPF_IN NPF_IPSecSA_Handle_t         saHandle);

/*
* Reserve SPI
*/

NPF_error_t NPF_IPSecInSA_ReserveSPI (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
            NPF_IN NPF_uint32_t                 selectorCount,
            NPF_IN NPF_IPSecSelector_t          *selectorArray);

/*
```

```
* Release SPI
*/

NPF_error_t NPF_IPSecInSA_ReleaseSPI (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
            NPF_IN NPF_uint32_t                 spi);


/*************************************************************************/
/* Misc. / Statistics Functions */
/*************************************************************************/

/*
* Policy Statistics
*/

NPF_error_t NPF_IPSecPolicy_GetStats (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
            NPF_IN NPF_boolean_t                resetStats);

/*
* SA Statistics
*/

NPF_error_t NPF_IPSecSA_GetStats (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
            NPF_IN NPF_IPSecSA_Handle_t         saHandle,
            NPF_IN NPF_boolean_t                resetStats);

/*
* Query ALL SPDs
*/

NPF_error_t NPF_IPSecQuery_AllSPDs (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting);

/*
* Query ALL SPD Bindings
*/

NPF_error_t NPF_IPSecQuery_AllSPDBindings (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecSPD_Handle_t        spdHandle);

/*
* Query ALL Policies
```

```
*/

NPF_error_t NPF_IPSecQuery_AllPolicies (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecSPD_Handle_t        spdHandle);

/*
* Query ALL Policy Bindings
*/

NPF_error_t NPF_IPSecQuery_AllPolicyBindings (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle);

/*
* Query ALL SAs
*/

NPF_error_t NPF_IPSecQuery_AllSAs (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecSPD_Handle_t        spdHandle);

/*
* Query Policy Data
*/

NPF_error_t NPF_IPSecQuery_PolicyData (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle);

/*
* Query SA Data
*/

NPF_error_t NPF_IPSecQuery_SAData (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecSA_Handle_t         saHandle);

/*
* Query SPD Handles
*/

NPF_error_t NPF_IPSecQuery_SPDHandle (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN_NPF_uint32_t                 countSPDIDs,
            NPF_IN NPF_IPSecSPD_ID_t            *spdIDArray);
```

```
/*
* Query Policy Handles
*/

NPF_error_t NPF_IPSecQuery_PolicyHandle (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_uint32_t                 countPolicyIDs
            NPF_IN NPF_IPSecPolicy_ID_t         *policyIDArray);


/*
* Query SA Handles
*/

NPF_error_t NPF_IPSecQuery_SAHandle (
            NPF_IN NPF_callbackHandle_t         cbHandle,
            NPF_IN NPF_correlator_t             cbCorrelator,
            NPF_IN NPF_errorReporting_t         errorReporting,
            NPF_IN NPF_IPSecPolicy_Handle_t     policyHandle,
            NPF_IN NPF_uint32_t                 countSAIDs,
            NPF_IN NPF_IPSecSA_ID_t             *arraySAIDs);

/*************************************************************************/
/********************* End Function Prototypes ***************************/
/*************************************************************************/

#ifdef __cplusplus
}
#endif
#endif /* __NPF_IPSEC_SAPI_H_ */
```

APPENDIX B ACKNOWLEDGEMENTS

Working Group Chair:

Vinoj Kumar, Agere Systems, vinoj@agere.com

Working Group Editor:

John Renwick, Agere Systems, jrenwick@agere.com

Task Group Chair(s):

Ho-Yen Chang, Ericsson, hoyen.chang@ericsson.com

Carsten Underbjerg, Ericsson, carsten.underbjerg@ericsson.com

The following individuals are acknowledged for their participation to IPsec TG teleconferences, plenary meetings, mailing list, and/or for their NPF contributions used for the development of this Implementation Agreement. This list may not be all-inclusive since only names supplied by member companies for inclusion here will be listed. The NPF wishes to thank all active participants to this Implementation Agreement, whether listed here or not.

The list is in alphabetical order of last names:

Nandagopal Ancha, Ericsson.

Magued Barsoum, EZchip.

Tim Bourne, Ericsson.

Rolf Christensen, Ericsson.

Anders Gammelgaard, Ericsson.

Ram Gopal, Nokia.

Zsolt Haraszti, Ericsson.

Karen Nielsen, Ericsson.

John Renwick, Agere.

Lene Sondergaard, Ericsson.

Soren Martin Sorensen, Ericsson

Xiaobing Zhang, Ericsson.

APPENDIX C LIST OF COMPANIES BELONGING TO NPF DURING APPROVAL PROCESS

Agere Systems
Altera
AMCC
Analog Devices
Avici
Cypress Semiconductor
Ericsson
Erlang Technology
ETRI
EZ Chip
Flextronics
FutureSoft
HCL Technologies
Hi/fn
IBM
IDT
Intel
IP Fabrics
IP Infusion
Kawasaki LSI
Motorola
Nokia
Nortel Networks
NTT Electronics
PMC-Sierra
Sun Microsystems
Teja Technologies
TranSwitch
U4EA Group
Xelerated
Xilinx
Zettacom