SSL Accelerator SAPI Implementation Agreement

August 7, 2006
Revision 1.0

Editor(s):

Suhail Ahmed, Intel, suhail.ahmed@intel.com

Asher Altman, Intel, asher.altman@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

Table of Figures

1 Revision History

Revision	Date	Reason for Changes
1.0	8/2/2006	Created 1.0 of this IA by applying editorial changes to NPF2005.068.02.

2 Acronyms and Abbreviations

The following acronyms and abbreviations are used in the specifications:

API	Applications Programming Interface
CP	Control Plane
FP	Forwarding Plane
FAPI	NPF Functional API
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LFB	Logical Functional Block
NE	Network Element
NP	Network Processor
NPE	Network Processing Element
NPF	Network Processing Forum
NPU	Network Processing Unit
openSSL	An open–source implementation of the SSL/TLS protocol. It is based on SSLeay. For more information about OpenSSL, see http://www.openssl.org/. openSSL is used here as an example application throughout the document.
RFC	Request For Comments (IETF standard)
SAPI	NPF Service API
SCTP	Stream Control Transmission Protocol
IM API	NPF Interface Management API
PH API	NPF Packet Handler API

3 References

NPF Implementation agreements	Revision	Description
NPF SAPI Conv.	1.0	Software API Conventions, Implementation Agreement
NPF Writing Conv.	Final	NPF Writing Conventions
NPF Lexicon	1.0	NPF API Framework Lexicon, Implementation Agreement
NPF Framework	1.0	NPF API Framework, Implementation Agreement
NPF IM API	Final + 0.3	NPF Interface Management API
NPF IPv4 UFwd	1.0	NPF Unicast Forwarding, Implementation Agreement
NPF Global		Software Global Definitions – to appear
NPF IPsec API	1.0	IPSec Service API

4 Introduction

Secure Socket Layer (SSL) has become a widely used mechanism for safeguarding the transmission and reception of Internet data. It is being used to secure online credit card transactions, Web page traffic, file and database information, and even email exchanges. A typical secure network platform will handle hundreds of megabits per second or more of secure packet bandwidth, and establish thousands of new Sessions each second. The computational complexity of such protocols requires that cryptographic hardware acceleration be used on platforms which process a large amount of secure network traffic. This document proposes a set of Services APIs (SAPI) for such hardware that offloads the SSL processing, namely the *SSL accelerator*.

During the SSL Session negotiation phase a cipher suite is selected and cryptography keys are exchanged. During the bulk data transfer phase the selected cipher suite is used to encrypt/decrypt the data. In addition, a cryptographic hash is used to validate that the data has not been corrupted. The Session negotiation phase uses public key cryptography to pass the keys between the client and server. Both the key exchange and the bulk cryptography are computationally expensive by design. Since most client machines are single user, the computational complexity does not generally represent a problem for the client machine. However, the server machine can easily become overwhelmed by the cryptographic computations because it may have to do these computations for hundreds or thousands of Sessions at a time.

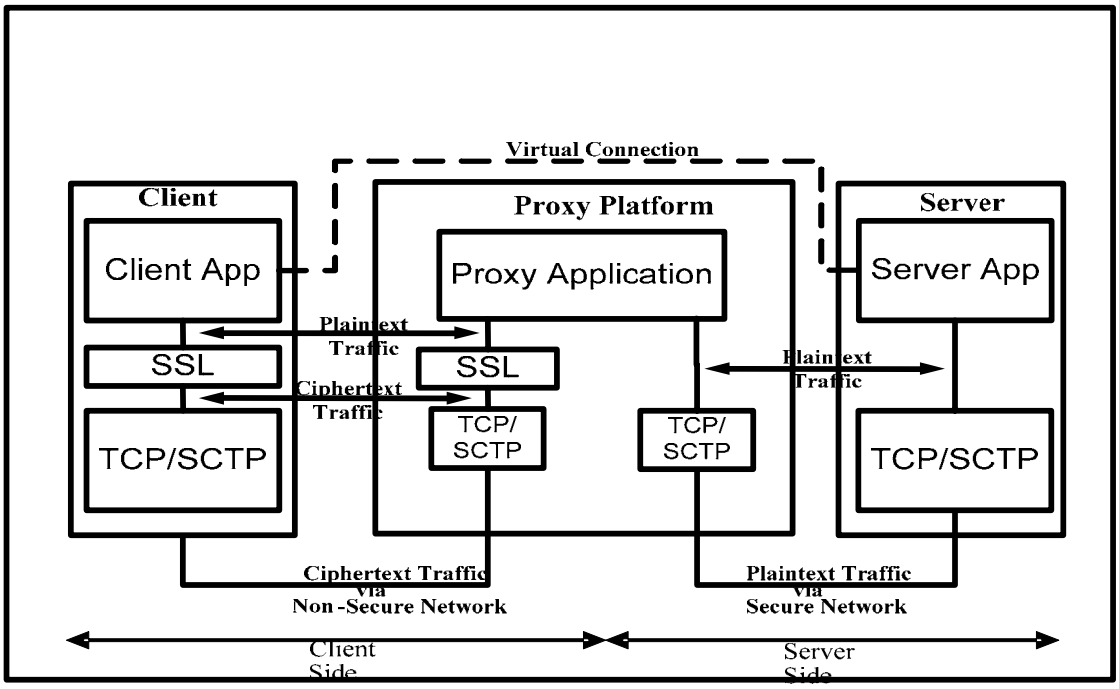

Figure 1: SSL Proxy for Acceleration

A number of acceleration strategies have been observed in the industry. One that is popular is the SSL acceleration proxy. The SSL processing is moved to a separate machine that offloads this processing from the server(s). The hardware platform for the SSL proxy can be optimized specifically for the task, so the cost per computational cycle is lower than for a compute server.

By using cryptographic acceleration operations in the traffic path near wire speed rates can be achieved rather than encrypting records on a general purpose server's CPU that contains no cryptographic acceleration.

Figure 1 above shows typical architectural blocks used in a SSL accelerator proxy. The side of the SSL Proxy that negotiates with SSL clients in the internet is the "client side". Similarly, the side of the SSL proxy that sets up connections to the target servers is the "server side". All bulk data records destined to the SSL accelerator from an external SSL client are termed as *inbound* records. Similarly all records originating from an openSSL[1] based application in the CP or target servers and going out from the interfaces of SSL accelerator are termed as *outbound* records.

SSL proxies may be hosted on a network element that performs other network or security functions such as load balancing and virus scanning.

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

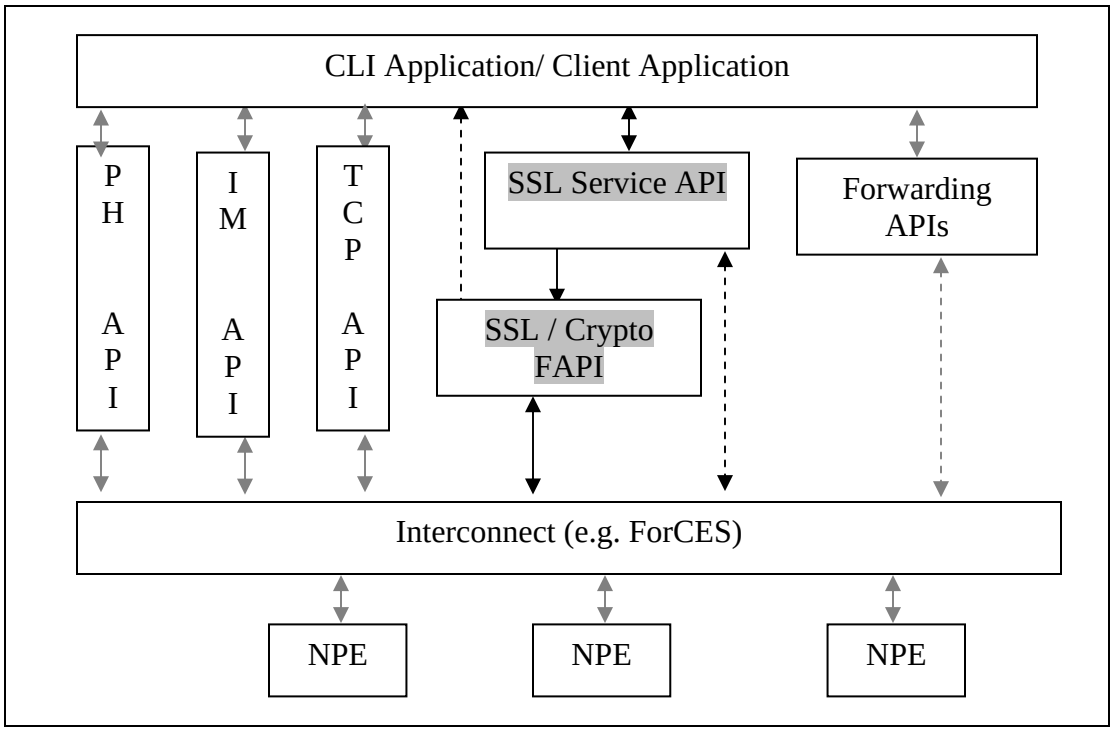

Figure 2: SSL - NPF APIs architectural relationship.

4.1 SSL Databases

Following SSL accelerator databases are defined for managing the SSL accelerator.

SSL Session Negotiation Database (SND): This database contains the Session negotiation parameters required to negotiate a successful SSL Session with SSL clients. Entries include:

[1] An open–source implementation of the SSL/TLS protocol. It is based on SSLeay. For more about OpenSSL, see http://www.openssl.org/. openSSL is used here as an example application.

proxy interface Address, Server Certificate, private key, cipher/digest, initial key, Client CA list etc. This database is looked up via the external proxy interface address.

SSL Session Cache Database (SSCD): This database is used to hold SSL Sessions that have been cached due to inactivity. Included in this database is all of the Session information that has to be retained in order to restore the Session to the SSL Session Database. Connections have their Session information moved from the SSD to the SSCD by using Get, Delete and Add SAPI calls. When decached, the Session information is moved back into the SSD using similar calls.

SSL Session Database (SSD): This database contains the state information associated with each SSL Session and is populated dynamically during the Session negotiation. Also, once the SSL Session is negotiated, a target (server) has to be selected (by querying into the TSD described below), and a connection to the target has to be made; this connection information is also part of this database. There is exactly one entry per client connection. The entries include: a session ID, keying material, bulk cipher used, digest used, client and target server connection identifiers etc. The session ID is assigned per client ←→ server connection. The size changes dynamically, as clients go up & down.

SSL Target Selector Database (TSD): This database is used to determine the target server address for a given Session. Entries include: target address along with the policy information, namely, proxy external address, layer 7 information (URL) or other proprietary classification information etc. The result of query into this database is a target server address to connect to. (Can be exploited for SSL VPN gateway)

Target selection may be performed using a number of different methods and is not the focus of this API. For instance, load balancing, as defined by the SLB SAPI may used as a target selection scheme. The TSD defined herein may be replaced with the SLB or other proprietary target selection methods.

The SSL Service API (SAPI) provides a generic interface for configuring and managing all or a subset of the above databases depending upon the actual distribution model of the SSL (see section 4.2 below). Furthermore, the SSL SAPI allows a client application to receive event notifications indicating packet drops, SSL alerts and other information. In accordance with the NPF framework model, the SSL SAPI is Network Element unaware.

4.2 SSL Distribution Models

SSL protocol comprises Session negotiation and bulk data processing. Depending upon the extent of the SSL functionality offload onto the Forwarding Planes there can be two main offload options: SSL Full offload and SSL Partial offload. The various SSL LFBs referred to in the following sections are as follows:

- SSL Session LFB: Responsible for processing the SSL Session negotiation with the SSL clients.
- SSL Bulk LFB: Responsible for bulk data processing i.e. de-cryption of inbound data and encryption of outbound data.
- SSL Record Layer LFB: Responsible for interacting with the TOE LFBs. It interprets SSL record headers in inbound data, recovers full records from TCP byte stream provided by TOE or other TCP protocol layer, directs records to appropriate LFB for processing,

The SSL LFBs maintain the context of the packet flow via the appropriate shared *metadata*. For example, the SSL proxy and bulk LFBs can share the session (or socket) identifier to identify a packet flow. Please refer to the SSL FAPI document (a separate NCI document that will capture the FAPI design details) for more details.

4.2.1 SSL Full offload

In this model, both the SSL Session and bulk are offloaded to the forwarding plane as shown in figure below. The SAPIs manage the Session Negotiation and the Target Selector Databases.

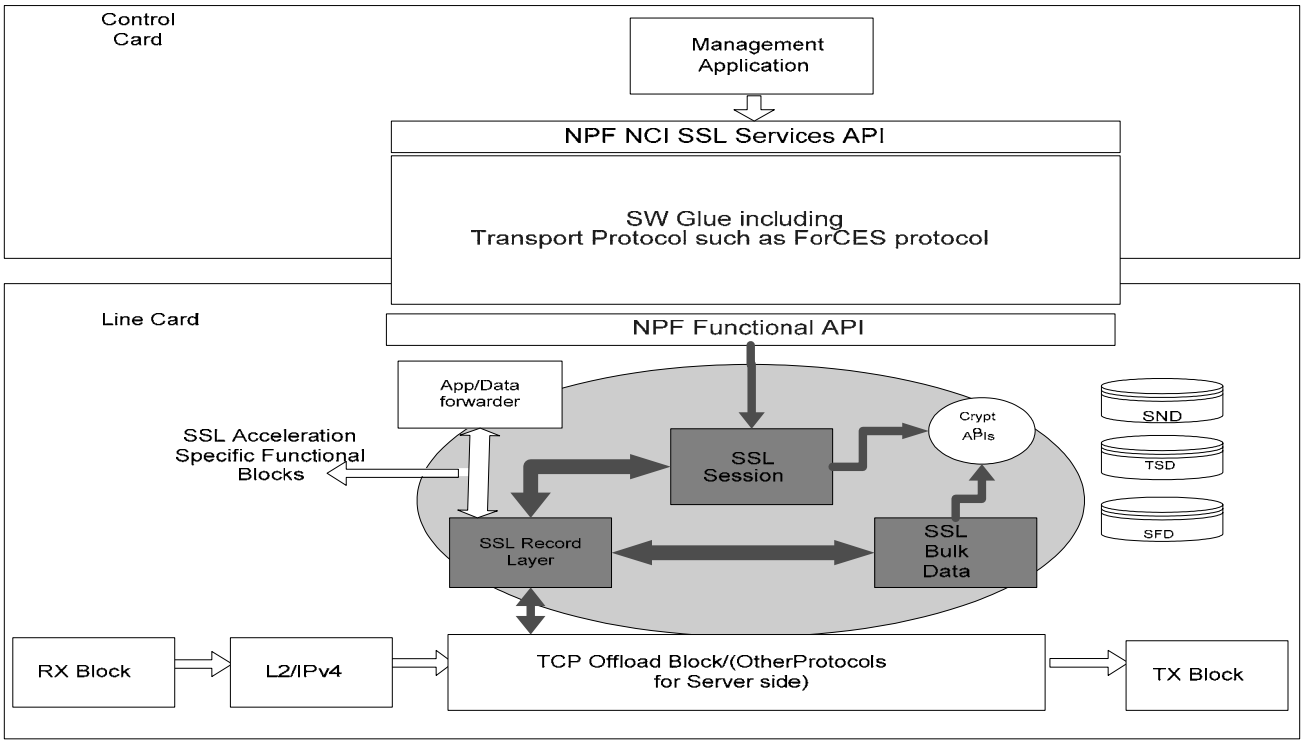

Figure 3: SSL Full Offload Model

4.2.1.1 Example packet flow

The Session traffic is handled in the forwarding plane. Typically the bulk data is also handled in the FP as well.

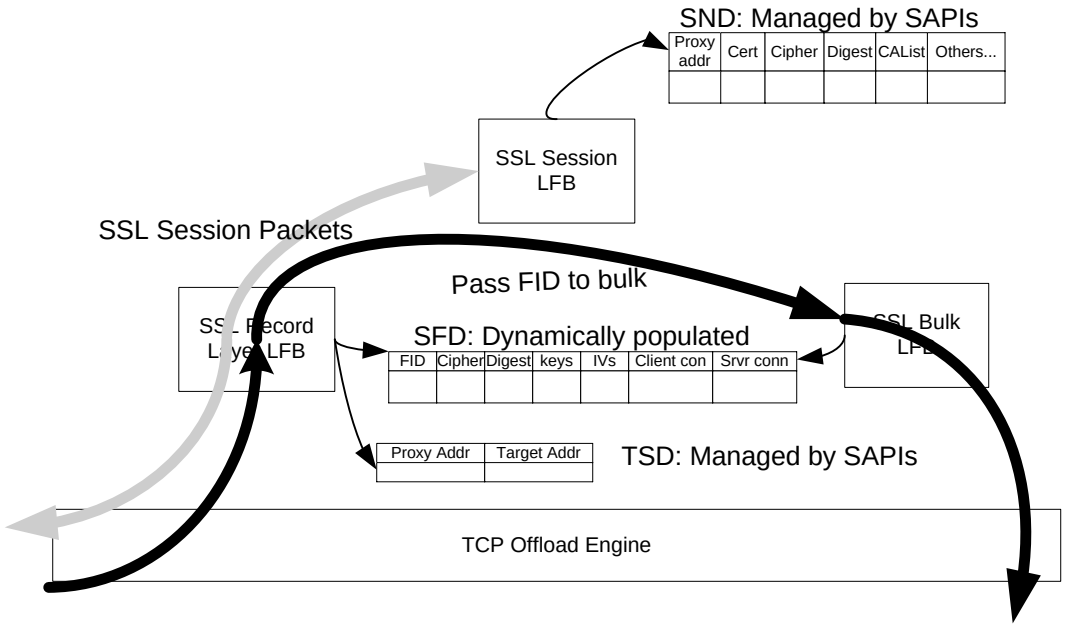

Figure 4: SSL packet flow for full offload model

Figure 4 above shows an example SSL packet flow for the full offload model. The light arrow shows represents the SSL Session records and the dark arrow represents the bulk data records. The SSL Record Layer LFB recovers SSL records from inbound TCP data and routes them to the SSL Bulk or SSL Session LFBs depending on their type. In case of inbound traffic, the records arrive encrypted and are decrypted in the Bulk LFB. The Bulk LFB also sends encrypted Session Records to the SSL Session LFB. (It is possible for the client application to register with the packet handler to receive the inbound bulk data, in which case the bulk data after being processed would be handed to the packet handler LFB).

4.2.2 SSL Partial offload

SSL partial offload model supports SSL Session negotiation in the CP and the SSL Bulk data processing in the FP. The following sub sections briefly outline the several partial offload models supported.

4.2.2.1 openSSL based client application *below* SAPI layer

In this model, the SSL Session resides in the control plane and bulk is offloaded to the forwarding plane as shown in figure below. As the openSSL based application is below the SAPI layer the SAPIs are exactly same as the SSL full offload. (The openSSL based application reads the Session negotiation information configured via the SAPIs). As the Session negotiation takes place in the CP, the SSD is managed via the FAPIs.

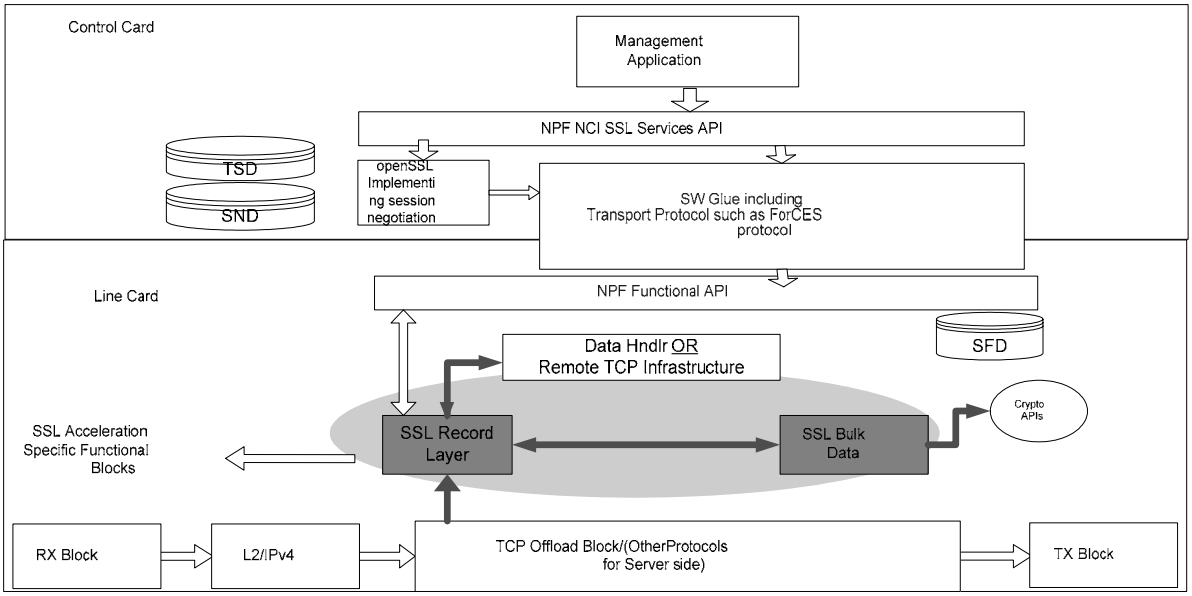

Figure 5: SSL Partial Offload Model – with openSSL based application below SAPI layer

4.2.2.2 openSSL based client application *above* SAPI layer

As the SSL Session negotiation processing is above the SAPI layer the SND does not need to be configured, however the SSL Session Database (SSD) has to be exposed at the SAPI level. Figure 6 below depicts such a model.

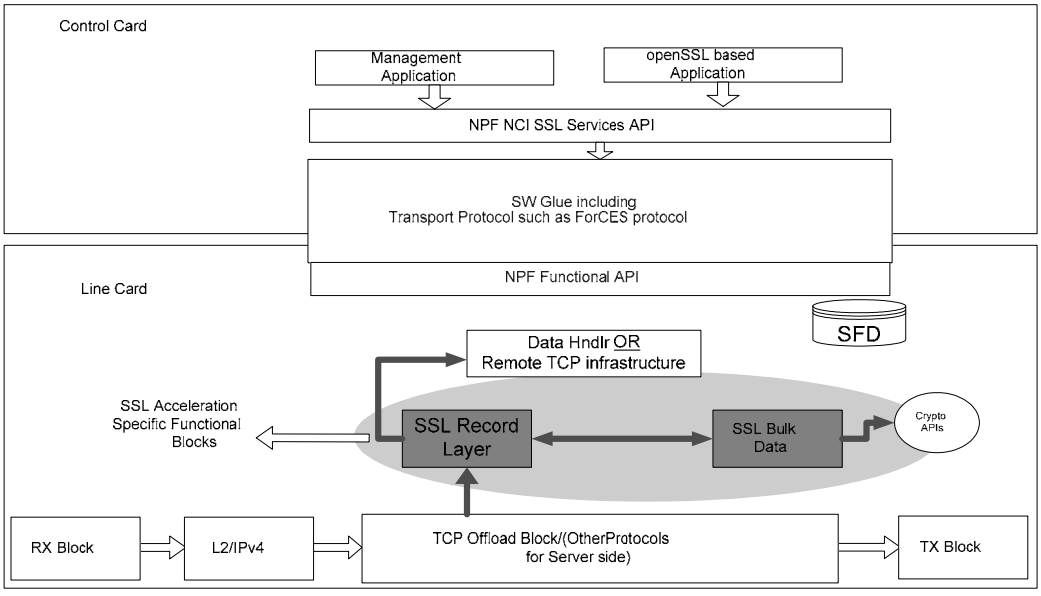

Figure 6: SSL Partial Offload Model – with openSSL based application above SAPI layer

4.2.2.3 Example packet flows

SSL Session records are processed in the CP. The SSL bulk packet flow can take two paths:

1. The bulk records are all processed, in the forwarding plane including encryption/de-cryption and sending records to/from client side & server side connections. The bulk data processing is similar to that of the full offload model (refer Figure 4). The only difference is that the SSD is managed via the FAPIs in this case during the Session negotiation.
2. All the inbound bulk records, after decrypting, are directed to the client application in the control plane and the client application redirects to the target side connection (or it may consume the records). Outbound bulk records are received by the client application in the CP which then requests the FP for encryption and sending out to the SSL client in the internet. Figure 7 below shows an example SSL packet flow for this partial offload model.

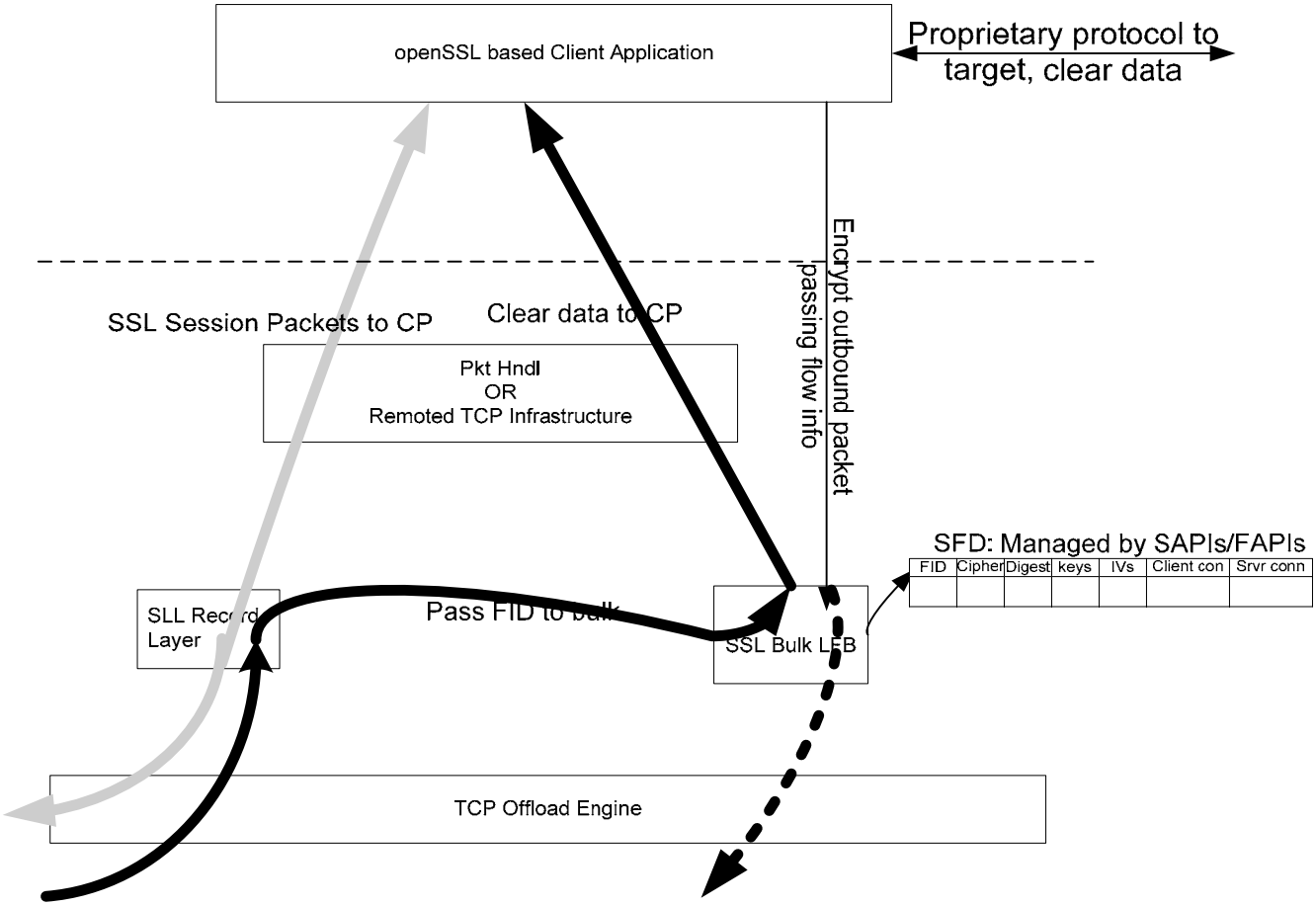

Figure 7: Packet flow for partial offload model with bulk data sent to CP

The light arrow represents the SSL Session records and the dark arrow represents the bulk data.records All the Session traffic flows to the application in the CP. Inbound bulk data is sent to the application in the CP. Packet flow to the CP is achieved via the packet handler (or the remoted TCP infrastructure[2]). The outbound traffic is sent via the *encrypt* SAPI call.

[2] It is possible that the TCP is *not* offloaded to the FP. In this case, all the records are received by to the application in the CP, which will invoke the SAPIs to get the records encrypted/decrypted.

4.3 Assumptions and External Requirements

- In the scope of this API, the structure and attributes of SSL reflect what is needed by the forwarding plane, not by the application. Applications are expected to maintain their own representation of the SSL databases.
- Memory allocation and usage model for this API implementation will be as dictated by the NPF Software Conventions Implementation Agreement.
- The SSL 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. SSL is no exception. For a usable system, the SSL SAPI is also reliant on the interface management API. Further, for partial offload models the applications will need the TCP APIs.
- SSL runs on top of either TCP or SCTP and the server side connection is TCP (or SCTP)
- There is one set of Session negotiation attributes per proxy (virtual) interface.
- Whenever TOE is assumed on the forwarding plane, it is fully operational including infrastructure to support TOE in the Control Plane. Thus, in case the client application resides in the control plane, (partial offload model), this infrastructure provides TCP packet flow into the control plane from forwarding plane. (In other words, it is assumed that a TCP application running on the CP can fully operate with TOE running on the FP)
- In case of the partial offload model, the application running on the control plane manages TCP connections both passive (client side) and active (server side).
- SSL design could leverage the Server Load Balancing (SLB) functional blocks for the selection of the target server for a particular SSL client request. This is essentially the SLB policy database as described in the SLB SAPI document. A simple case of this SSL target selector database (TSD) is described here for full operation of SSL in absence of SLB. In this case, a more elaborate proprietary target selection mechanism can be used by extending the example TSD.

4.4 Scope

The following are in scope of this document:

1. Complete SSL Services APIs that enable full design and implementation SSL Logical Functional blocks (LFBs). This implies that the packet flow options and the three SSL distribution models are fully supported.
2. The SSL 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.

Following are considered out of scope of this document:

1. SSL VPN gateway, but the overall design of SAPIs tries to accommodate any future VPN gateway efforts
2. Proprietary protocols can exist between the accelerator and the target servers and is considered out of scope of this document.

3. This document is not a design document; for elaborate details of the forwarding plane based LFBs, SSL Accelerator FAPI document needs to be referred (future NCI work).

4.5 Dependencies

The SSL SAPI utilizes conventions from the following NPF documents.

[NPF Lexicon]
[NPF SAPI conv]
[NPF Framework]
[NPF IM API]
[NPF PH API]
[NPF Global]

5 Data Types

5.1 SSL Handle Types

```
typedef NPF_uint32_t    NPF_NCI_SSLSessionNegHandle_t;
```
This is a 32-bit value that is used to identify an SSL Session Negotiation Database (SND).

```
typedef NPF_uint32_t    NPF_NCI_SSLSessionHandle_t;
```
This is a 32-bit value that is used to identify an SSL Session Database (SSD).

```
typedef NPF_uint32_t    NPF_NCI_SSLSessionCacheHandle_t;
```
This is a 32-bit value that is used to identify an SSL Session Cache Database (SSCD).

```
typedef NPF_uint32_t    NPF_NCI_SSLTargetSelHandle_t;
```
This is a 32-bit value that is used to identify an SSL Target Selector Database (TSD).

```
typedef NPF_uint32_t    NPF_NCI_SSLSessionId_t;
```
This is a 32-bit value that is used to identify a client Session

```
typedef NPF_uint32_t    NPF_NCI_SSLSessionCacheId_t;
```
This is a 32-bit value that is used to identify a cached client Session

5.2 SSL Constants

```
/*Constants that define the SSL Session attributes sizes*/
#define MAX_INITIAL_KEY_SIZE              256
#define MAX_SERVER_CERTIFICATE_SIZE       1024
#define MAX_CLIENT_CA_LIST_SIZE           128

/*Constants that define the bulk data processing information sizes*/
#define MAX_SERVER_CIPHER_KEY_SIZE        4
#define MAX_CLIENT_CIPHER_KEY_SIZE        4
#define MAX_SERVER_MAC_KEY_SIZE           4
#define MAX_CLIENT_MAC_KEY_SIZE           4

/*Size of a url inside the clear data, used to select the target server*/
#define MAX_URL_SIZE                      1024
```

5.3 SSL Generic IP Address

```
/*
*     Single IP address
*/


typedef union
```

```
{
      NPF_IPv4Address_t v4;               /* IPv4 address */
      NPF_IPv6Address_t v6;               /* IPv6 address */
} NPF_NCI_SSLIPAddress_t



/*
*     SSL Proxy Local address and port info
*/


typedef struct
{
      /* IP Version */
      /* IPv4 = 4, IPv6 = 6 */
      NPF_uint8_t       IP_Version;
      union {
            NPF_IPv4Address_t v4;               /* IPv4 address */
            NPF_IPv6Address_t v6;               /* IPv6 address */
            } addr;
      /* Transport Protocol */
      /* TCP =1 or SCTP = 2 */
      NPF_uint8_t       L4_Protocol;
      union {
         NPF_uint16_t   TCP_Port;
         NPF_uint32_t   SCTP_Port;
            } port;

} NPF_NCI_SSLAddressPortInfo_t;
```

5.4 SSL Cipher and Digests

```
/*
*     The NPF_NCI_SSLCipherAndDigest enum defines the various SSL ciphers and
      digests that can be supported by the SSL accelerator.
*/
typedef enum
{
      NPF_NCI_RSA_WITH_NULL_MD5 = 1,
      NPF_NCI_RSA_WITH_NULL_SHA,
      NPF_NCI_RSA_EXPORT_WITH_RC4_40_MD5,
      NPF_NCI_RSA_WITH_RC4_128_MD5,
      NPF_NCI_RSA_WITH_RC4_128_SHA,
      NPF_NCI_RSA_EXPORT_WITH_RC2_CBC_40_MD5,
      NPF_NCI_RSA_WITH_IDEA_CBC_SHA,
      NPF_NCI_RSA_EXPORT_WITH_DES40_CBC_SHA,
      NPF_NCI_RSA_WITH_DES_CBC_SHA,
      NPF_NCI_RSA_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_DH_DSS_EXPORT_WITH_DES40_CBC_SHA,
      NPF_NCI_DH_DSS_WITH_DES_CBC_SHA,
      NPF_NCI_DH_DSS_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_DH_RSA_EXPORT_WITH_DES40_CBC_SHA,
      NPF_NCI_DH_RSA_WITH_DES_CBC_SHA,
      NPF_NCI_DH_RSA_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA,
```

```
      NPF_NCI_DHE_DSS_WITH_DES_CBC_SHA,
      NPF_NCI_DHE_DSS_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA,
      NPF_NCI_DHE_RSA_WITH_DES_CBC_SHA,
      NPF_NCI_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_DH_anon_EXPORT_WITH_RC4_40_MD5,
      NPF_NCI_DH_anon_WITH_RC4_128_MD5,
      NPF_NCI_DH_anon_EXPORT_WITH_DES40_CBC_SHA,
      NPF_NCI_DH_anon_WITH_DES_CBC_SHA,
      NPF_NCI_DH_anon_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_RSA_EXPORT1024_WITH_DES_CBC_SHA = 98,
      NPF_NCI_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA,
      NPF_NCI_RSA_EXPORT1024_WITH_RC4_56_SHA,
      NPF_NCI_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA,
      NPF_NCI_DHE_WITH_RC4_128_SHA,

      /* Experimental Algorithm Identifiers */

      NPF_NCI_START_OF_EXPERIMENTAL_SECTION=500,

      /* Proprietary Algorithm Identifiers */

      NPF_NCI_START_OF_PROPRIETARY_SECTION=1000
} NPF_NCI_SSLCipherAndDigest;
```

NOTE: If there are more ciphers/digests they need to be accommodated as well.

5.5 SSL Databases

5.5.1 SSL Session negotiation database (SND)

The SND contains the Session negotiation parameters associated with an SSL proxy (external) interface. These parameters are required to successfully negotiate a Session with an SSL client.

```
/*The following structure represents an entry in the SSL negotiation
Database*/

typedef struct {
   /* Address/Port Info */
   NPF_NCI_SSLAddressPortInfo_t     proxyAddress;

   /*cipher/digest supported*/
   NPF_NCI_SSLCipherAndDigest cipher_and_digest;

   /*Suggested Initial key*/
   NPF_uint8_t                initialKey[MAX_INITIAL_KEY_SIZE];

   /*Server's Public Certificate */
   NPF_uint8_t                certificate[MAX_SERVER_CERTIFICATE_SIZE];

   /*Server's Private Key */
   NPF_uint32_t               *privateKey;

   /*Comma separated, List of trusted CA's*/
   NPF_uint8_t                clientCAList[MAX_CLIENT_CA_LIST_SIZE];

} NPF_NCI_SSLSessionNegParams_t;
```

Note: There could be more fields added to the above SND structure as needed.

5.5.1.1 NPF_NCI_SSLSessionNeg_Array_t

```
typedef struct
{
      NPF_uint32_t nCount;
      NPF_NCI_SSLSessionNegParams_t* sndArray;
} NPF_NCI_SSLSessionNeg_Array_t;
```

5.5.2 SSL Target Selector Database (TSD)

This database provides the target server address to connect to based on proprietary classification methods. Target selection may be performed using a number of different methods and is not the focus of this API. For instance, load balancing, as defined by the SLB SAPI may used as a target selection scheme. The following database represents an example of a basic target selection scheme. The TSD defined herein may be replaced with the SLB or other proprietary target selection methods.

```
/*Target selection policy type*/
typedef enum
{
      NPF_NCI_SSL_POLICY_PROXY_ADDR,      /*Indicates that the target is
                                          selected via the proxy address*/
      NPF_NCI_SSL_POLICY_URL              /*Indicates that the target is
                                          selected via the url in the data*/
} NPF_NCI_SSLPolicyType_t;

/*Target selection policy. An example of proxy external address and url in
the data is shown below*/

typedef struct {
   NPF_NCI_SSLPolicyType_t    policy_type;      /* policy type*/
   union
   {
      NPF_NCI_SSLIPAddress_t  proxyAddress;     /*External interface addr*/
      NPF_uint8_t             url[MAX_URL_SIZE];/*Url in clear data*/

      /*Other Proprietary fields go here */
   }u;
} NPF_NCI_SSLPolicy_t;

typedef struct {
   NPF_uint8_t                IP_Version;       /* IPv4 = 4, IPv6 = 6 */
   NPF_NCI_SSLIPAddress_t     targetAddress;    /* Target addr*/
   NPF_NCI_SSLPolicy_t        policy;
} NPF_NCI_SSLTargetSelectorParams_t;
```

This database represents the target selection in the simplest form, namely, one target server per proxy interface address. The `policy` field can be exploited to support more granular classification, like the *url* information contained in the decrypted data. Further, more elaborate classification information for supporting the SSL VPN gateway can be also easily supported.

Once the target addressing information is determined, the SSL accelerator can re-route the clear data via a TCP connection for SSL acceleration or any other proprietary mechanism.

5.5.2.1 NPF_NCI_SSLTargetSelector_Array_t

```
typedef struct {
      NPF_uint32_t nCount;
      NPF_NCI_SSLTargetSelectorParams_t *tsdArray;
} NPF_NCI_SSLTargetSelector_Array_t;
```

5.5.3 SSL Session Database (SSD)

This database contains per SSL Session information, viz, the encryption/de-cryption cipher & digest keys for a Session etc.

The following structure contains the SSL client and the target server connection mapping:

```
typedef struct {
   NPF_uint32_t   clientConnID;     /*Identifies the client connection*/
   NPF_uint32_t   targetConnID;     /*Identifies the target connection*/
} NPF_NCI_SSLConnectionParams_t;

typedef struct {
   NPF_uint32_t               serverKeyInfo[MAX_SERVER_CIPHER_KEY_SIZE];
   NPF_uint32_t               clientKeyInfo[MAX_CLIENT_CIPHER_KEY_SIZE];
   NPF_uint32_t               serverMACKeyInfo[MAX_SERVER_MAC_KEY_SIZE];
   NPF_uint32_t               clientMACKeyInfo[MAX_CLIENT_MAC_KEY_SIZE];
   NPF_NCI_SSLCipherAndDigest       cipherAndDigest;
   NPF_NCI_SSLConnectionParams_t    connectionMapping; /*SSL Client<-->target
                                                         mapping*/
} NPF_NCI_SSLSessionParams_t;
```

This database is critical for bulk data processing. It is used by the SSL bulk LFB to select the encryption/decryption information. The entries are populated dynamically during SSL Session negotiation with an SSL client. In bound Records are looked up by the `clientConnID` for proper decryption, similarly the outbound packets are looked up by the `targetConnID` for proper encryption. This database has to be managed via SAPIs only for the partial offload case where the openSSL based application is above the SAPI layer (see section 4.2.2.2)

5.5.3.1 NPF_NCI_SSLSession_Array_t

```
typedef struct {
      NPF_uint32_t                  nCount;
      NPF_NCI_SSLSessionParams_t    *sessionArray;
} NPF_NCI_SSLSession_Array_t;
```

5.5.4 SSL Session Cache Database (SSCD)

This database contains the SSL Session cache information that is populated as a result of inactivity on an SSL Session.

The following structure contains the SSL client and the target server connection mapping:

```
typedef struct {
   NPF_uin32_t clientConnID;        /*Identifies the client connection*/
   NPF_uin32_t targetConnID;        /*Identifies the target connection*/
} NPF_NCI_SSLConnectionParams_t;

typedef struct {
   NPF_uint32_t              serverKeyInfo[MAX_SERVER_CIPHER_KEY_SIZE];
   NPF_uint32_t              clientKeyInfo[MAX_CLIENT_CIPHER_KEY_SIZE];
   NPF_uint32_t              serverMACKeyInfo[MAX_SERVER_MAC_KEY_SIZE];
   NPF_uint32_t              clientMACKeyInfo[MAX_CLIENT_MAC_KEY_SIZE];
   NPF_NCI_SSLCipherAndDigest       cipherAndDigest;
   NPF_NCI_SSLConnectionParams_t    connectionMapping; /*SSL Client<-->target
                                                         mapping*/
   NPF_uint32_t              sessionCacheTimeoutValue; /* in seconds */
} NPF_NCI_SSLSessionCacheParams_t;
```

5.5.4.1 NPF_NCI_SSLSessionCache_Array_t

```
typedef struct {
      NPF_uint32_t                  nCount;
      NPF_NCI_SSLSessionCacheParams_t     *sessionCacheArray;
} NPF_NCI_SSLSessionCache_Array_t;
```

5.5.5 NPF_NCI_SSLData_t

This structure contains the SSL bulk data (encrypted/decrypted) information.

```
typedef struct {
      NPF_uint32_t                  data_len;
      NPF_uint8_t                   *data;
} NPF_NCI_SSLData_t;
```

5.6 SSL Statistics

Two basic structures provide the SSL statistics as follows:

```
/*
      Crypto related stats
*/
typedef struct {
    NPF_uint32_t  HMAC_Errors;       /* HMAC (hash) errors -inbound only- */
    NPF_uint32_t  decryptErrors;     /* Decryption  errors -inbound*/
    NPF_uint32_t  encryptErrors;     /* Decryption  errors -outbound*/
    NPF_uint32_t  replayErrors;      /* Replay attacks -inbound only- */
  } NPF_NCI_SSLCryptoStats_t;

typedef struct {
      NPF_uint8_t             IP_Version;       /* IPv4 = 4, IPv6 = 6 */
      NPF_NCI_SSLIPAddress_t  clientAddr;
      NPF_NCI_SSLIPAddress_t  proxyAddr;
      NPF_NCI_SSLIPAddress_t  targetAddr;
}NPF_NCI_SSL_ConnectionInfo_t;

/*
      Connection related stats
*/
typedef struct {
  NPF_uint32_t                      numSessions;
  NPF_NCI_SSL_ConnectionInfo_t      *sslEndpoints;
} NPF_NCI_SSLConnectionStats_t;
```

5.7 Error Codes

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

/*
*     SSL reserved error codes in relation to other NPF APIs
*     Note** The maximum range is 100
*/
#define NPF_NCI_SSL_BASE_ERR 700 /* Base value of 700 wrt other NPF codes */
```

5.7.1 SSL Error Type : NPF_NCI_SSLErrorType_t

```
/*
*     SSL Error Type
*/
typedef NPF_uint32_t NPF_NCI_SSLErrorType_t;  /* SSL Error Type */
```

5.7.2 Generic Error Codes

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

```
/* The Interface handle provided was not recognized as being valid */
#define NPF_NCI_SSL_E_INVALID_IF_HANDLE \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + 1)

/* Already registered */
#define NPF_NCI_SSL_E_ALREADY_REGISTERED \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + 2)

/* Optional feature not supported */
#define NPF_NCI_SSL_E_OPTIONAL_FEATURE_NOT_SUPPORTED \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + 3)

/* System was unable to allocate sufficient memory to complete operation */
#define NPF_NCI_SSL_E_NOMEMORY \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + 4)
```

5.7.3 Specific Error Codes

```
/* Invalid SND handle */
#define NPF_NCI_SSL_E_INVALID_SND_HANDLE \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
      NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 1)

/* Invalid SSD handle */
#define NPF_NCI_SSL_E_INVALID_SSD_HANDLE \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
      NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 2)

/* Invalid TSD handle */
#define NPF_NCI_SSL_E_INVALID_TSD_HANDLE \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 3)

/* Invalid cipher algorithm */
#define NPF_NCI_SSL_E_INVALID_ENC_ALGO \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
      NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 4)

/* Invalid cipher algorithm keylen */
#define NPF_SSL_E_INVALID_ENC_ALGO_KEYLEN \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
      NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 5)

/* Invalid digest algorithm */
#define NPF_NCI_SSL_E_INVALID_DIGEST_ALGO \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
      NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 6)

/* Invalid digest algorithm keylen */
#define NPF_NCI_SSL_E_INVALID_DIGEST_ALGO_KEYLEN \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
       NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 7)

/* Invalid Initialization vector*/
#define NPF_NCI_SSL_E_INVALID_IV \
```

```
((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
 NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 8)
```

5.8 Data Structures for Completion Callbacks

A completion callback is defined for each of the functions in this API.

5.8.1 Asynchronous Response

This is the asynchronous response structure, which is part of the CallbackData structure that is passed to the caller in the asynchronous response. It contains an error/success code, and a function-specific structure embedded in a union.

```
typedef struct {
      NPF_NCI_SSLErrorType_t error;
      union {
            NPF_NCI_SSLSessionId_t              sslSessionId;
            NPF_NCI_SSLSessionCacheId_t         sslSessionCacheId;
            NPF_NCI_SSLSessionNegHandle_t       sndHandle;
            NPF_NCI_SSLTargetSelHandle_t        tsdHandle;
            NPF_NCI_SSLSessionHandle_t          ssdHandle;
            NPF_NCI_SSLSessionCacheHandle_t     sscdHandle;
            NPF_NCI_SSLSessionNeg_Array_t       sndArray;
            NPF_NCI_SSLTargetSelector_Array_t   tddArray;
            NPF_NCI_SSLSession_Array_t          ssdArray;
            NPF_NCI_SSLSessionCache_Array_t     sscdArray;
            NPF_NCI_SSLConnectionStats_t        sslConnStats;
            NPF_NCI_SSLCryptoStats_t            sslCryptoStats;
            NPF_NCI_SSLData_t                   ssl_data;
      } u;
} NPF_NCI_SSLAsyncResponse_t;
```

5.8.2 Callback Type

The callback response contains one of the following codes, indicating the function that was called to cause the callback. This code tells the application how to interpret the data included in the union that is part of the response structure.

```
/* completion callback types */
typedef enum NPF_NCI_SSLCallbackType {
      NPF_NCI_SSL_SESSION_NEGOTIATION_DATABASE_CREATE  = 1,
      NPF_NCI_SSL_SESSION_NEGOTIATION_DATABASE_DELETE  = 2,
      NPF_NCI_SSL_SESSION_NEGOTIATION_INFO_ADD         = 3,
      NPF_NCI_SSL_SESSION_NEGOTIATION_INFO_DELETE      = 4,
      NPF_NCI_SSL_SESSION_NEGOTIATION_INFO_PURGE       = 5,
      NPF_NCI_SSL_SESSION_NEGOTIATION_DATABASE_QUERY   = 6,
      NPF_NCI_SSL_CONNECTION_STATISTICS_QUERY          = 7,
      NPF_NCI_SSL_CRYPTO_STATISTICS_QUERY              = 8,
      NPF_NCI_SSL_TARGET_SELECTOR_DATABASE_CREATE      = 9,
      NPF_NCI_SSL_TARGET_SELECTOR_DATABASE_DELETE      = 10,
      NPF_NCI_SSL_TARGET_SELECTOR_INFO_ADD             = 11,
      NPF_NCI_SSL_TARGET_SELECTOR_INFO_DELETE          = 12,
      NPF_NCI_SSL_TARGET_SELECTOR_INFO_PURGE           = 13,
      NPF_NCI_SSL_TARGET_SELECTOR_DATABASE_QUERY       = 14,
      NPF_NCI_SSL_SESSION_DATABASE_CREATE              = 15,
      NPF_NCI_SSL_SESSION_DATABASE_DELETE              = 16,
      NPF_NCI_SSL_SESSION_INFO_ADD                     = 17,
```

```
        NPF_NCI_SSL_SESSION_INFO_GET                    = 18,
        NPF_NCI_SSL_SESSION_INFO_DELETE                 = 19,
        NPF_NCI_SSL_SESSION_INFO_PURGE                  = 20,
        NPF_NCI_SSL_SESSION_DATABASE_QUERY              = 21,
        NPF_NCI_SSL_SESSION_CACHE_DATABASE_CREATE       = 22,
        NPF_NCI_SSL_SESSION_CACHE_DATABASE_DELETE       = 23,
        NPF_NCI_SSL_SESSION_CACHE_INFO_ADD              = 24,
        NPF_NCI_SSL_SESSION_CACHE_INFO_GET              = 25,
        NPF_NCI_SSL_SESSION_CACHE_INFO_DELETE           = 26,
        NPF_NCI_SSL_SESSION_CACHE_INFO_PURGE            = 27,
        NPF_NCI_SSL_SESSION_CACHE_DATABASE_QUERY        = 28,
        NPF_NCI_SSL_BULK_ENCRYPT_SEND                   = 29,
        NPF_NCI_SSL_BULK_ENCRYPT                        = 30,
        NPF_NCI_SSL_BULK_DECRYPT                        = 31
    } NPF_NCI_SSLCallbackType_t;
```

5.8.3 Callback Data

This is the callback response structure, which is passed to the caller in the asynchronous response from a function call. It contains the callback type that identifies the function called, the allOK flag, the number of responses and an array of response structures depending on the call.

```
    typedef struct {
        NPF_NCI_SSLCallbackType_t     type;
        NPF_boolean_t                 allOK;
        NPF_uint32_t                  numResp;
        NPF_NCI_SSLAsyncResponse_t   *resp;
    } NPF_NCI_SSLCallbackData_t;
```

5.9 Data Structures for Event Notifications

The following sections detail the information related to SSL events. When an event notification routine is invoked, one of the parameters will be a structure of information related to one or more events.

5.9.1 Event Notification Types

The event type indicates the type of event data in the union of event structures returned in `NPF_NCI_SSLEventData_t`.

```
/*
*     SSL Event
*/
typedef enum
{
      NPF_NCI_SSL_REPLAY_PACKET        = 1,/* Replay packet caught */
      NPF_NCI_SSL_DECRYPTION_FAILED    = 2,/* SSL decryption failed */
      NPF_NCI_SSL_ENCRYPTION_FAILED    = 3,/* SSL encryption failed */
      NPF_NCI_SSL_NO_TARGET_MAPPING    = 4,/* No target server found*/
      NPF_NCI_SSL_CLIENT_AUTH_FAILED   = 5,/* Client authentication failed*/
      NPF_NCI_SSL_ALERT_RECEIVED       = 6,/* Received an SSL Alert*/
      NPF_NCI_SSL_MEM_FULL             = 7 /*A crypto NPU ran out of memory*/
} NPF_NCI_SSLEvent_t;

#define SSL_MAX_EVENT_TYPES 7
```

Note: More event types can be added in future as needed.

5.9.2 SSL event bitmap : NPF_NCI_SSLEventMask_t

```
/*
*     SSL event bitmask used in the event registration call.
*/
typedef NPF_uint32_t NPF_NCI_SSLEventMask_t;

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

#define NPF_NCI_SSL_EVENT_ALL_DISABLE                (0) /* disable all */
#define NPF_NCI_SSL_REPLAY_PACKET_ENABLE             (1 << 1)
#define NPF_NCI_SSL_DECRYPTION_FAILED_ENABLE         (1 << 2)
#define NPF_NCI_SSL_ENCRYPTION_FAILED_ENABLE         (1 << 3)
#define NPF_NCI_SSL_NO_TARGET_MAPPING_ENABLE         (1 << 4)
#define NPF_NCI_SSL_CLIENT_AUTH_FAILED_ENABLE        (1 << 5)
#define NPF_NCI_SSL_ALERT_RECEIVED_ENABLE            (1 << 6)
#define NPF_NCI_SSL_MEM_FULL_ENABLE                  (1 << 7)
#define NPF_SSL_EVENT_ALL_ENABLE                     0xFFFFFFFF
```

5.9.3 Event Notification Structures

This section describes the various events which MAY be implemented.

It is important to note that even if an implementation does not support any of these events, the implementation still needs to provide the register and deregister event function to enable interoperability.

This structure defines all the possible event definitions for SSL API. An event type field indicates which member of the union is relevant in the specific structure.

5.9.3.1 Information on packet of interest: NPF_NCI_SSLPacketInfo_t

```
/*Problem packet*/
typedef struct
{
      NPF_uint8_t             IP_Version; /* 4 = IPv4, 6 = IPv6 */
      NPF_uint8_t             protocol;   /* IP Transport Protocol */
      NPF_NCI_SSLIPAddress_t  clientAddr; /*Address on client*/
      NPF_NCI_SSLIPAddress_t  proxyAddr;  /*Address on proxy interface*/
      NPF_uint16_t            srcPort;    /* IP Source Port */
      NPF_uint16_t            alertVal;   /* SSL alert value, 200 for others*/
} NPF_NCI_SSLPacketInfo_t;
```

For each of the events define above, NPF_NCI_SSLPacketDropped_t will be appropriately populated.

5.9.4 SSL Event Data : NPF_NCI_SSLEventData_t

```
/*
*     SSL Event Data
*/
typedef struct
{
      NPF_NCI_SSLEvent_t eventType;
      NPF_NCI_SSLPacketInfo_t packet;
} NPF_NCI_SSLEventData_t;
```

5.9.5 SSL Event Data : NPF_NCI_SSLEvent_Array_t

```
/*
*     SSL Event Array
*/
typedef struct
{
      NPF_uint16_t            n_data;           /*Number of events in array*/
      NPF_NCI_SSLEventData_t  *eventDataArray;  /* Array of event
                                                notifications */
} NPF_NCI_SSLEventArray_t;
```

6 Functions

6.1 Completion Callback

This callback function is for the application to register an asynchronous response handling routine to the SSL API implementation.

6.1.1 Completion Callback Function Signature

```
typedef void (*NPF_NCI_SSLCallbackFunc_t)(
      NPF_IN NPF_userContext_t            userContext,
      NPF_IN NPF_correlator_t             correlator,
      NPF_IN NPF_NCI_SSLCallbackData_t    sslCallbackData);
```

Description

This callback function is for the application to register asynchronous response handling routine to the NPF SSL API implementation. This callback function is intended to be implemented by the application, and be registered to the NPF SSL API implementation through `NPF_NCI_sslRegister()` 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 SSL API function call was made. The correlator is used by the application mainly to distinguish between multiple invocations of the same function.
- `sslCallbackData`
  Response information related to the SSL API function call. Contains information that are common among all functions, as well as information that are specific to a particular function. See `NPF_NCI_SSLCallbackData_t` definition for details.

Output Parameters

None.

Return Value

None.

6.2 Event Notification Function Calls

This event notification function is for the application to register an event handler routine to the SSL API implementation.

6.2.1 NPF_NCI_SSLEventCallFunc_t

```
typedef void (*NPF_NCI_SSLEventCallFunc_t) (
   NPF_IN NPF_userContext_t       userContext,
   NPF_IN NPF_NCI_SSLEventArray_t data);
```

Description

This function is a registered event notification routine for handling SSL API events.

Input Parameters

- `userContext` - The context item that was supplied by the application when the event callback routine was registered.
- `data` – A structure containing an array of event data structures and a count to indicate how many events are present. Each of these `NPF_NCI_SSLEventData_t` members contains event specific information and a type field to identify the particular event.

Output Parameters

None

Return Value

None

6.3 Callback Registration/Deregistration Function Calls

This section defines the registration and de-registration functions used to install and remove an asynchronous response callback routine.

6.3.1 Completion Callback Registration Function

```
NPF_error_t NPF_NCI_SSLRegister(
      NPF_IN NPF_userContext_t                  userContext,
      NPF_IN NPF_NCI_SSLCallbackFunc_t          sslCallbackFunc,
      NPF_OUT NPF_callbackHandle_t  *sslCallbackHandle);
```

Description

This function is used by an application to register its completion callback function for receiving asynchronous responses related to NPF SSL API function calls. Application may register multiple callback functions using this function. The callback function is identified by the pair of userContext and sslCallbackFunc, and for each individual pair, a unique `sslCallbackHandle` will be assigned for future reference. Since the callback function is identified by both userContext and sslCallbackFunc, duplicate registration of same callback function with different userContext is allowed. Also, same userContext can be shared among different callback functions. Duplicate registration of the same userContext and sslCallbackFunc pair has no effect, and will output a handle that is already assigned to the pair, and will return `NPF_E_CALLBACK_ALREADY_REGISTERED`.

Note : `NPF_NCI_SSLRegister( )` 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 SSL API implementation.
- `sslCallbackFunc`
  The pointer to the completion callback function to be registered.

Output Parameters

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

Return Values

- `NPF_NO_ERROR`
  The registration completed successfully.
- `NPF_E_BAD_CALLBACK_FUNCTION`

`sslCallbackFunc` is NULL.

- `NPF_E_CALLBACK_ALREADY_REGISTERED`
  No new registration was made since the `userContext` and `sslCallbackFunc` pair was already registered.
  Note: Whether this should be treated as an error or not is dependent on the application.

6.3.2 Completion Callback Deregistration

```
NPF_error_t NPF_NCI_SSLDeregister(
      NPF_IN NPF_callbackHandle_t   sslCallbackHandle);
```

Description

This function is used by an 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_NCI_sslDeregister`( ) is a synchronous function and has no completion callback associated with it.

Input Parameters

- `sslCallbackHandle`
  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 effect to the registered callback functions.

6.4 Event Registration/Deregistration Function Calls

This section defines the registration and de-registration functions used to install and remove an event handler routine

6.4.1 NPF_NCI_SSLEventRegister

```
NPF_error_t NPF_NCI_SSLEventRegister(
   NPF_IN NPF_userContext_t               userContext,
   NPF_IN NPF_NCI_SSLEventCallFunc_t      eventCallFunc,
   NPF_IN NPF_NCI_SSLEventMask_t          eventMask,
   NPF_OUT NPF_callbackHandle_t           *eventCallHandle);
```

Description

This function is used by an application to register its event handling routine for receiving notifications of SSL 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_E_CALLBACK_ALREADY_REGISTERED`.

Input Parameters

- `userContext` – A context item for uniquely identifying the context of the application registering the event handling routine. The exact value will be provided back to the registered event handling routine as its first parameter when it is called. Applications can assign any value to the `userContext` and the value is completely opaque to the SSL API implementation
- `eventCallFunc` – The pointer to the event handling routine to be registered.
- `eventMask` – This is a bit mask of the SSL events. It allows the application to register for those selected events.

Output Parameters

- `eventCallHandle` - A unique identifier assigned for the registered `userContext` and `eventCallFunc` pair. This handle will be used when deregistering the `userContext` and `eventCallFunc` pair.

Return Values

- `NPF_NO_ERROR` - The registration completed successfully.
- `NPF_E_BAD_CALLBACK_FUNCTION` – The `eventCallFunc` is NULL, or otherwise invalid.
- `NPF_E_CALLBACK_ALREADY_REGISTERED` – No new registration was made since the `userContext` and `eventCallFunc` pair was already registered.

- `NPF_NCI_SSL_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

6.4.2 NPF_NCI_SSLEventDeregister

```
NPF_error_t NPF_NCI_SSLEventDeregister(
   NPF_IN NPF_callbackHandle_t eventCallHandle);
```

Description

This function is used by an application to de-register an event handler routine which was previously registered to receive notifications of SSL events. It represents a unique user context and event handling routine pair.

Input Parameters

- `eventCallHandle` - The unique identifier returned to the application when the event callback routine was registered.

Output Parameters

None

Return Values

- `NPF_NO_ERROR` – The de-registration completed successfully.
- `NPF_E_BAD_CALLBACK_HANDLE` – The de-registration did not complete successfully due to problems with the callback handle provided.

6.5 Event definition signature

NPF SSL implementations can generate the events listed in section 5.9.1, type `NPF_NCI_SSLEvent_t`.

6.6 Completion Callbacks and Error Returns

Each of the functions defined in the SSL 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_E_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.

6.7 SSL Service APIs (SSL SAPIs)

The SSL SAPIs defined in the sections below support the three SSL distribution models described earlier in section 4.2:

- SSL SAPIs that manage the SSL Session negotiation database (sections 6.7.2 - 6.7.7) along with the target selector SAPIs (sections 6.7.27 - 6.7.32) are required to manage SSL accelerators based on the SSL full offload model described in section 4.2.1 and SSL partial offloaded model described in section 4.2.2.
- To support the SSL partial offload model described in section 4.2.2.2, a different set of SAPIs are defined (sections 6.7.8 - 6.7.14) that manage the SSL bulk database.

6.7.1 Generic Description of SSL Functions

Syntax

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

Description

The SSL 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.
- plus any further parameters.

Output Parameters

Describe the output parameters for each function. In the SSL 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_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_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.

6.7.2 NPF_NCI_SSLSessionNegotiationDatabaseCreate()

```
NPF_error_t    NPF_NCI_SSLSessionNegotiationDatabaseCreate(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired);
```

Description

This function creates the SSL Session Negotiation Database and returns a handle associated with the SND. The handle is used to perform any subsequent operations on this SND. On success, an implementation provided SND handle is returned via the asynchronous callback.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_NEGOTIATION_DATABASE_CREATE` is generated in response to this function call. This contains the return codes for any errors encountered. If the call is successful an SND handle is returned. The SND 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_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.

6.7.3 NPF_NCI_SSLSessionNegotiationDatabaseDelete()

```
NPF_error_t    NPF_NCI_SSLSessionNegotiationDatabaseDelete(
               NPF_IN   NPF_callbackHandle_t           cbHandle,
               NPF_IN   NPF_correlator_t               correlator,
               NPF_IN   NPF_errorReporting_t           cbDesired,
               NPF_IN   NPF_NCI_SSLSessionNegHandle_t  sndHandle);
```

Description

This function administratively destroys the SND, which has been previously created by the `NPF_NCI_SSLSessionNegotiationDatabaseCreate()` function. Before destroying the SND, an implicit SND Flush operation is performed in order to clean the SND.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sndHandle`: The SND handle to be deleted

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_NEGOTIATION_DATABASE_DELETE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SND_HANDLE` – The Received SND handle was not recognized as being valid.

6.7.4 NPF_NCI_SSLSessionNegotiationInfoAdd()

```
NPF_error_t    NPF_NCI_SSLSessionNegotiationInfoAdd(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionNegHandle_t sndHandle,
                  NPF_IN      NPF_uint32_t            numEntries,
                  NPF_IN      NPF_NCI_SSLSessionNegParams_t
                                                      *sessionArray);
```

Description

This function is used to add an array of Session negotiation parameter entries to the SSL Session negotiation database identified by `sndHandle`.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sndHandle`: Handle to the SND.
- `numEntries`: Number of entries in the sessionArray
- `sessionArray`: Array of `NPF_NCI_SSLSessionNegParams_t`

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_NEGOTIATION_INFO_ADD` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SND_HANDLE` – The Received SND handle was not recognized as being valid.

6.7.5 NPF_NCI_SSLSessionNegotiationInfoDelete()

```
NPF_error_t    NPF_NCI_SSLSeSessionNegotiationInfoDelete(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionNegHandle_t sndHandle,
                  NPF_IN      NPF_uint32_t            numEntries,
                  NPF_IN      NPF_NCI_SSLSessionNegParams_t
                                                      *sessionArray);
```

Description

This function is used to delete an array of SSL Session negotiation entries in the SSL Session negotiation database identified by the `sndHandle`. In the case of any existing entry in SND, the Session negotiation parameters are updated with the new ones.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_sslRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sndHandle`: The SSL Session negotiation database handle.
- `numEntries`: Number of entries in the sessionArray
- `sessionArray`: Array of Session negotiation parameters

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_NEGOTIATION_INFO_DELETE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SND_HANDLE` – The Received SND handle was not recognized as being valid.

6.7.6 NPF_NCI_SSLSessionNegotiationInfoPurge()

```
NPF_error_t    NPF_NCI_SSLSessionNegotiationInfoPurge(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionNegHandle_t
                              sndHandle);
```

Description

This function is used to purge the SSL Session Negotiation Database identified by the `sndHandle`.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sndHandle`: The handle to SND to be purged.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_NEGOTIATION_INFO_PURGE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SND_HANDLE` – The Received SND handle was not recognized as being valid.

6.7.7 NPF_NCI_SSLSessionNegotiationDatabaseQuery()

```
NPF_error_t    NPF_NCI_SSLSessionNegotiationDatabaseQuery(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionNegHandle_t sndHandle);
```

Description

This function is used to query the SND identified by the `sndHandle` handle.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sndHandle`: Handle to the SND.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

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

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SND_HANDLE` – The Received SND handle was not recognized as being valid.

6.7.8 NPF_NCI_SSLSessionDatabaseCreate()

```
NPF_error_t    NPF_NCI_SSLSessionDatabaseCreate(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired);
```

Description

This function creates the SSL Session database (SSD). The handle is used to perform any subsequent operations on this SSD. On success, an implementation provided SSD handle is returned via the asynchronous callback.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_DATABASE_CREATE` is generated in response to this function call. This contains the return codes for any errors encountered. If the call is successful, then a SSD handle is returned. The SSD 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_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.

6.7.9 NPF_NCI_SSLSessionDatabaseDelete()

```
NPF_error_t    NPF_NCI_SSLSessionDatabaseDelete(
               NPF_IN   NPF_callbackHandle_t          cbHandle,
               NPF_IN   NPF_correlator_t              correlator,
               NPF_IN   NPF_errorReporting_t          cbDesired,
               NPF_IN   NPF_NCI_SSLSessionHandle_t    ssdHandle);
```

Description

This function administratively destroys the SSD, which have been previously created by the `NPF_NCI_SSLSessionDatabaseCreate()` function. Before destroying the SSD, an implicit SSD Flush operation is performed in order to clean the SSD.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `ssdHandle`: The associated SSD handle

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_DATABASE_DELETE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSD_HANDLE` – The received SSD handle was not recognized as being valid.

6.7.10 NPF_NCI_SSLSessionInfoAdd()

```
NPF_error_t    NPF_NCI_SSLSessionInfoAdd(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionHandle_t    ssdHandle,
                  NPF_IN      NPF_NCI_SSLSessionParams_t
                                                      *ssdEntry);
```

Description

This function is used to add a Session entry to the SSD identified by `ssdHandle`. This entry is made dynamically during the SSL Session negotiation.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `ssdHandle`: Associated handle to the SSD.
- `ssdEntry`: `NPF_NCI_SSLSessionParams_t` entry.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_INFO_ADD` is generated in response to this function call. A single asynchronous response, `NPF_NCI_SSLAsyncResponse_t`, will be passed to the callback function containing the Session identifier. The following error codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSD_HANDLE` – The received SSD handle was not recognized as being valid.

6.7.11 NPF_NCI_SSLSessionInfoGet()

```
NPF_error_t    NPF_NCI_SSLSessionInfoGet(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionHandle_t    ssdHandle,
                  NPF_IN      NPF_NCI_SSLSessionParams_t
                                                      *ssdEntry,
                  NPF_OUT     NPF_NCI_SSLSessionParams_t
                                                      *ssdEntryOutput);
```

Description

This function is used to get a Session entry from the SSD identified by `ssdHandle`. This is used to get a session entry to cache in the session cache database.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `ssdHandle`: Associated handle to the SSD.
- `ssdEntry`: `NPF_NCI_SSLSessionParams_t` entry to be output

Output Parameters

- `ssdEntryOutput`. `NPF_NCI_SSLSessionParams_t` entry to be output data

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_INFO_GET` is generated in response to this function call. A single asynchronous response, `NPF_NCI_SSLAsyncResponse_t`, will be passed to the callback function containing the Session identifier. The following error codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSD_HANDLE` – The received SSD handle was not recognized as being valid.

6.7.12 NPF_NCI_SSLSessionInfoDelete()

```
NPF_error_t    NPF_NCI_SSLSessionInfoDelete(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionHandle_t ssdHandle,
                  NPF_IN      NPF_uint32_t            numEntries,
                  NPF_IN      NPF_NCI_SSLSessionId_t
                                                      *ssdArray);
```

Description

This function is used to delete an array of SSL Session Database entries in the SSD Handle. In the case of any existing entry in SSD, the connection related parameters are updated with the new ones.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_sslRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `ssdHandle`: Associated SSD handle.
- `numEntries`: Number of entries in the ssdArray
- `ssdArray`: Array of SSL session entries

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_INFO_DELETE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSD_HANDLE` – The received SSD handle was not recognized as being valid.

6.7.13 NPF_NCI_SSLSessionInfoPurge()

```
NPF_error_t    NPF_NCI_SSLSessionInfoPurge(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionHandle_t ssdHandle);
```

Description

This function is used to purge the SSL Session Database identified by the `ssdHandle`.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `ssdHandle`: The handle to SSD to be purged.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_INFO_PURGE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSD_HANDLE` – The received SSD handle was not recognized as being valid.

6.7.14 NPF_NCI_SSLSessionDatabaseQuery()

```
NPF_error_t    NPF_NCI_SSLSessionDatabaseQuery(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionHandle_t    ssdHandle);
```

Description

This function is used to query the SSD identified by the `ssdHandle`.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `ssdHandle`: Handle to the SSD.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

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

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSD_HANDLE` – The received SSD handle was not recognized as being valid.

6.7.15 NPF_NCI_SSLSessionCacheDatabaseCreate()

```
NPF_error_t    NPF_NCI_SSLSessionCacheDatabaseCreate(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired);
```

Description

This function creates the SSL Session Cache database (SSCD). The handle is used to perform any subsequent operations on this SSCD. On success, an implementation provided SSCD handle is returned via the asynchronous callback.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_CACHE_DATABASE_CREATE` is generated in response to this function call. This contains the return codes for any errors encountered. If the call is successful, then a SSCD handle is returned. The SSCD 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_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.

6.7.16 NPF_NCI_SSLSessionCacheDatabaseDelete()

```
      NPF_error_t    NPF_NCI_SSLSessionCacheDatabaseDelete(
                     NPF_IN   NPF_callbackHandle_t           cbHandle,
                     NPF_IN   NPF_correlator_t               correlator,
                     NPF_IN   NPF_errorReporting_t           cbDesired,
                     NPF_IN   NPF_NCI_SSLSessionCacheHandle_t
sscdHandle);
```

Description

This function administratively destroys the SSCD, which have been previously created by the `NPF_NCI_SSLSessionCacheDatabaseCreate()` function. Before destroying the SSCD, an implicit SSCD Flush operation is performed in order to clean the SSCD.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sscdHandle`: The associated SSCD handle

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_CACHE_DATABASE_DELETE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSCD_HANDLE` – The received SSCD handle was not recognized as being valid.

6.7.17 NPF_NCI_SSLSessionCacheInfoAdd()

```
   NPF_error_t    NPF_NCI_SSLSessionCacheInfoAdd(
                     NPF_IN      NPF_callbackHandle_t    cbHandle,
                     NPF_IN      NPF_correlator_t        correlator,
                     NPF_IN      NPF_errorReporting_t    cbDesired,
                     NPF_IN      NPF_NCI_SSLSessionCacheHandle_t
sscdHandle,
                     NPF_IN      NPF_NCI_SSLSessionCacheParams_t
                                                         *sscdEntry);
```

Description

This function is used to add a Session Cache entry to the SSCD identified by `sscdHandle`. This entry is made dynamically during the SSL Session Cache negotiation.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sscdHandle`: Associated handle to the SSCD.
- `sscdEntry`: `NPF_NCI_SSLSessionCacheParams_t` entry.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_CACHE_INFO_ADD` is generated in response to this function call. A single asynchronous response, `NPF_NCI_SSLAsyncResponse_t`, will be passed to the callback function containing the Session Cache identifier. The following error codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSCD_HANDLE` – The received SSCD handle was not recognized as being valid.

6.7.18 NPF_NCI_SSLSessionCacheInfoGet()

```
NPF_error_t    NPF_NCI_SSLSessionCacheInfoGet(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionCacheHandle_t
                                                      sscdHandle,
                  NPF_IN      NPF_NCI_SSLSessionCacheParams_t
                                                      *sscdEntry,
                  NPF_OUT     NPF_NCI_SSLSessionCacheParams_t
                                                   *sscdEntryOutput);
```

Description

This function is used to get a Session Cache entry from the SSCD identified by `sscdHandle`. This is used to get a cached entry to restore to the session database..

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sscdHandle`: Associated handle to the SSCD.
- `sscdEntry`: `NPF_NCI_SSLSessionCacheParams_t` entry to be output

Output Parameters

- sscdEntryOutput. `NPF_NCI_SSLSessionCacheParams_t` entry to be output data

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_CACHE_INFO_GET` is generated in response to this function call. A single asynchronous response, `NPF_NCI_SSLAsyncResponse_t`, will be passed to the callback function containing the Session Cache identifier. The following error codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSCD_HANDLE` – The received SSCD handle was not recognized as being valid.

6.7.19 NPF_NCI_SSLSessionCacheInfoDelete()

```
      NPF_error_t    NPF_NCI_SSLSessionCacheInfoDelete(
                        NPF_IN      NPF_callbackHandle_t    cbHandle,
                        NPF_IN      NPF_correlator_t        correlator,
                        NPF_IN      NPF_errorReporting_t    cbDesired,
                        NPF_IN      NPF_NCI_SSLSessionCacheHandle_t
sscdHandle,
                        NPF_IN      NPF_uint32_t            numEntries,
                        NPF_IN      NPF_NCI_SSLSessionCacheId_t
                                                            *sscdArray);
```

Description

This function is used to delete an array of SSL Session Cache Database entries in the SSCD Handle. In the case of any existing entry in SSCD, the connection related parameters are updated with the new ones.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_sslRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sscdHandle`: Associated SSCD handle.
- `numEntries`: Number of entries in the sscdArray
- `sscdArray`: Array of SSL session entries

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_CACHE_INFO_DELETE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSCD_HANDLE` – The received SSCD handle was not recognized as being valid.

6.7.20 NPF_NCI_SSLSessionCacheInfoPurge()

```
      NPF_error_t    NPF_NCI_SSLSessionCacheInfoPurge(
                        NPF_IN      NPF_callbackHandle_t    cbHandle,
                        NPF_IN      NPF_correlator_t        correlator,
                        NPF_IN      NPF_errorReporting_t    cbDesired,
                        NPF_IN      NPF_NCI_SSLSessionCacheHandle_t
sscdHandle);
```

Description

This function is used to purge the SSL Session Cache Database identified by the `sscdHandle`.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sscdHandle`: The handle to SSCD to be purged.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_SESSION_CACHE_INFO_PURGE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSCD_HANDLE` – The received SSCD handle was not recognized as being valid.

6.7.21 NPF_NCI_SSLSessionCacheDatabaseQuery()

```
    NPF_error_t    NPF_NCI_SSLSessionCacheDatabaseQuery(
                      NPF_IN      NPF_callbackHandle_t    cbHandle,
                      NPF_IN      NPF_correlator_t        correlator,
                      NPF_IN      NPF_errorReporting_t    cbDesired,
                      NPF_IN      NPF_NCI_SSLSessionCacheHandle_t
sscdHandle);
```

Description

This function is used to query the SSCD identified by the `sscdHandle`.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sscdHandle`: Handle to the SSCD.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Asynchronous Callback Response

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

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSCD_HANDLE` – The received SSCD handle was not recognized as being valid.

6.7.22 NPF_NCI_SSLConnStatisticsQuery()

```
NPF_error_t    NPF_NCI_SSLConnStatisticsQuery(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionNegHandle_t sndHandle);
```

Description

This function is used query the connection SSL statistics identified by the SND handle.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sndHandle`: The handle to SND.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Additionally, the following may also be returned.

Asynchronous Callback Response

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

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SND_HANDLE` – The Received SND handle was not recognized as being valid.

6.7.23 NPF_NCI_SSLCryptoStatisticsQuery()

```
NPF_error_t    NPF_NCI_SSLCryptoStatisticsQuery (
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionNegHandle_t sndHandle);
```

Description

This function is used query the crypto SSL statistics identified by the SND handle.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `sndHandle`: The handle to SND.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Additionally, the following may also be returned.

Asynchronous Callback Response

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

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SND_HANDLE` – The Received SND handle was not recognized as being valid.

6.7.24 NPF_NCI_SSLBulkEncryptSend()

```
NPF_error_t    NPF_NCI_SSLBulkEncryptSend(
                  NPF_IN      NPF_callbackHandle_t          cbHandle,
                  NPF_IN      NPF_correlator_t              correlator,
                  NPF_IN      NPF_errorReporting_t          cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionHandle_t    ssdHandle,
                  NPF_IN      NPF_uint8_t                   *data,
                  NPF_IN      NPF_NCI_SSLSessionId_t  sslSession
                  );
```

Description

This function is used request data to be encrypted and is sent out on flow specified by `sslSession`.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `ssdHandle`: Handle to the SSD.
- `data`: Data to be encrypted
- `sslSession`: Identifies what encryption to be used and the SSL client connection identifier.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Additionally, the following may also be returned.

Asynchronous Callback Response

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

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSD_HANDLE` – The received SSD handle was not recognized as being valid.

Additional Discussion

This API is invoked by the client application only for the outbound traffic to be encrypted before being sent to the SSL client in the internet.

6.7.25 NPF_NCI_SSLBulkEncrypt()

```
NPF_error_t    NPF_NCI_SSLBulkEncrypt(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionHandle_t    ssdHandle,
                  NPF_IN      NPF_uint8_t             *data,
                  NPF_IN      NPF_NCI_SSLSessionId_t  sslSession
                  );
```

Description

This function is used request data to be encrypted as specified by `sslSession`.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `ssdHandle`: Handle to the SSD.
- `data`: Data to be encrypted
- `sslSession`: Identifies what encryption to be used and the SSL client connection identifier.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Additionally, the following may also be returned.

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_BULK_ENCRYPT` is generated in response to this function call. A single asynchronous response, `NPF_NCI_SSLAsyncResponse_t`, will be passed to the callback function containing the requested information. The callback will contain the encrypted data information.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSD_HANDLE` – The received SSD handle was not recognized as being valid.

Additional Discussion

This API can be used by the client application for the non-TOE partial offload model.

6.7.26 NPF_NCI_SSLBulkDecrypt()

```
NPF_error_t    NPF_NCI_SSLBulkDecrypt(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionHandle_t    ssdHandle,
                  NPF_IN      NPF_uint8_t             *data,
                  NPF_IN      NPF_NCI_SSLSessionId_t  sslSession
                  );
```

Description

This function is used request data to be encrypted as specified by `sslSession`.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `ssdHandle`: Handle to the SSD.
- `data`: Data to be decrypted
- `sslSession`: Identifies what decryption to be used and the SSL client connection identifier.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Additionally, the following may also be returned.

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_BULK_DECRYPT` is generated in response to this function call. A single asynchronous response, `NPF_NCI_SSLAsyncResponse_t`, will be passed to the callback function containing the requested information. The callback will contain the decrypted data information.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_SSD_HANDLE` – The received SSD handle was not recognized as being valid.

Additional Discussion

This API can be used by the client application for the non-TOE partial offload model.

6.7.27 NPF_NCI_SSLTargetSelectorDatabaseCreate()

```
NPF_error_t    NPF_NCI_SSLTargetSelectorDatabaseCreate(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired
                  );
```

Description

This function creates the SSL target selector database (TSD). The handle is used to perform any subsequent operations on this TSD. On success, an implementation provided TSD handle is returned via the asynchronous callback.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Additionally, the following may also be returned.

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_TARGET_SELECTOR_DATABASE_CREATE` is generated in response to this function call. This contains the return codes for any errors encountered. If the call is successful, then a TSD handle is returned. The TSD 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_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.

6.7.28 NPF_NCI_SSLTargetSelectorDatabaseDelete()

```
NPF_error_t    NPF_NCI_SSLTargetSelectorDatabaseDelete(
               NPF_IN   NPF_callbackHandle_t          cbHandle,
               NPF_IN   NPF_correlator_t              correlator,
               NPF_IN   NPF_errorReporting_t          cbDesired,
               NPF_IN   NPF_NCI_SSLTargetSelHandle_t  tsdHandle);
```

Description

This function administratively destroys the TSD, which have been previously created by the `NPF_NCI_SSLTargetSelectorDatabaseCreate()` function. Before destroying the TSD, an implicit TSD Flush operation is performed in order to clean the TSD.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `tsdHandle`: The TSDD handle to be deleted

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Additionally, the following may also be returned.

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_TARGET_SELECTOR_DATABASE_DELETE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_TSD_HANDLE` – The Received TSD handle was not recognized as being valid.

6.7.29 NPF_NCI_SSLTargetSelectorInfoAdd()

```
NPF_error_t    NPF_NCI_SSLTargetSelectorInfoAdd(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLTargetSelHandle_t  tsdHandle,
                  NPF_IN      NPF_uint32_t            numEntries,
                  NPF_IN      NPF_NCI_SSLTargetSelectorParams_t
                                                      *tsdArray);
```

Description

This function is used to add an array of target selector entries to the TSD identified by `tsdHandle`.

Input Parameters

- `cbHandle`: The callback handle returned by `NPF_NCI_SSLRegister()` call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `tsdHandle`: Handle to the TSD.
- `numEntries`: Number of entries in the tsdArray
- `tsdArray`: Array of NPF_NCI_SSLTargetSelectorParams_t

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Additionally, the following may also be returned.

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_TARGET_SELECTOR_INFO_ADD` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_TSD_HANDLE` – The received TSD handle was not recognized as being valid

6.7.30 NPF_NCI_SSLTargetSelectorInfoDelete()

```
NPF_error_t    NPF_NCI_SSLTargetSelectorInfoDelete(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLTargetSelHandle_t tsdHandle,
                  NPF_IN      NPF_uint32_t            numEntries,
                  NPF_IN      NPF_NCI_SSLTargetSelectorParams_t
                                                      *tsdArray);
```

Description

This function is used to delete an array of SSL target entries in the TSD. In the case of any existing entry in TSD, the target related parameters are updated with the new ones.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_sslRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `tsdHandle`: The TSD handle.
- `numEntries`: Number of entries in the tsdArray
- `tsdArray`: Array of Target Selector entries

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Additionally, the following may also be returned.

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_TARGET_SELECTOR_INFO_DELETE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_TSD_HANDLE` – The received TSD handle was not recognized as being valid

6.7.31 NPF_NCI_SSLTargetSelectorInfoPurge()

```
NPF_error_t    NPF_NCI_SSLTargetSelectorInfoPurge(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLTargetSelHandle_t tsdHandle);
```

Description

This function is used to purge the SSL Target selector Database identified by the `tsdHandle`.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_SSLRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `tsdHandle`: The handle to TSD to be purged.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Additionally, the following may also be returned.

Asynchronous Callback Response

A callback of type `NPF_NCI_SSL_TARGET_SELECTOR_INFO_PURGE` is generated in response to this function call. This contains the return codes for any errors encountered. The following codes may be returned.

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_TSD_HANDLE` – The received TSD handle was not recognized as being valid

6.7.32 NPF_NCI_SSLTargetSelectorDatabaseQuery()

```
NPF_error_t    NPF_NCI_SSLTargetSelectorDatabaseQuery(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLTargetSelHandle_t  tsdHandle);
```

Description

This function is used to query the TSD identified by the `tsdHandle`.

Input Parameters

- `cbHandle`: The callback handle returned by NPF_NCI_sslRegister() call.
- `correlator`: A 32-bit value that will be returned in the callback for this function call.
- `cbDesired`: The desired level of callback verbosity: always, never, or only upon error.
- `tsdHandle`: Handle to the TSD.

Output Parameters

None.

Synchronous Return Codes

As defined in the Synchronous Return Codes [Generic Description of SSL Functions].

Additionally, the following may also be returned.

Asynchronous Callback Response

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

- `NPF_NO_ERROR` – The operation was successful.
- `NPF_NCI_SSL_E_NOMEMORY` – The system was unable to allocate sufficient memory to complete this operation.
- `NPF_NCI_SSL_E_INVALID_TSD_HANDLE` – The received TSD handle was not recognized as being valid.

7 API Call and Event Capabilities

These tables are included as a summary for informative purposes.

7.1 API Functions

API function Name	Function Required
NPF_NCI_SSLSessionNegotiationDatabaseCreate()	Required
NPF_NCI_SSLSessionNegotiationDatabaseDelete()	Required
NPF_NCI_SSLSessionNegotiationInfoAdd()	Required
NPF_NCI_SSLSessionNegotiationInfoDelete()	Required
NPF_NCI_SSLSessionNegotiationInfoPurge()	Required
NPF_NCI_SSLSessionNegotiationDatabaseQuery()	Required
NPF_NCI_SSLSessionDatabaseCreate()	Optional
NPF_NCI_SSLSessionDatabaseDelete()	Optional
NPF_NCI_SSLSessionInfoAdd()	Optional
NPF_NCI_SSLSessionInfoGet()	Optional
NPF_NCI_SSLSessionInfoDelete()	Optional
NPF_NCI_SSLSessionInfoPurge()	Optional
NPF_NCI_SSLSessionCacheDatabaseCreate()	Optional
NPF_NCI_SSLSessionCacheDatabaseDelete()	Optional
NPF_NCI_SSLSessionCacheInfoAdd()	Optional
NPF_NCI_SSLSessionCacheInfoGet()	Optional
NPF_NCI_SSLSessionCacheInfoDelete()	Optional
NPF_NCI_SSLSessionCacheInfoPurge()	Optional
NPF_NCI_SSLConnStatisticsQuery()	Optional
NPF_NCI_SSLCryptoStatisticsQuery()	Optional
NPF_NCI_SSLBulkEncryptSend()	Optional
NPF_NCI_SSLBulkEncrypt()	Optional
NPF_NCI_SSLBulkDecrypt()	Optional
NPF_NCI_SSLTargetSelectorDatabaseCreate()	Required
NPF_NCI_SSLTargetSelectorDatabaseDelete()	Required
NPF_NCI_SSLTargetSelectorInfoAdd()	Required
NPF_NCI_SSLTargetSelectorInfoDelete()	Required
NPF_NCI_SSLTargetSelectorInfoPurge()	Required
NPF_NCI_SSLTargetSelectorDatabaseQuery()	Required

7.2 API Events

Event Name	Event Required
NPF_NCI_SSL_REPLAY_PACKET	Optional
NPF_NCI_SSL_DECRYPTION_FAILED	Optional
NPF_NCI_SSL_ENCRYPTION_FAILED	Optional
NPF_NCI_SSL_NO_TARGET_MAPPING	Optional
NPF_NCI_SSL_CLIENT_AUTH_FAILED	Optional
NPF_NCI_SSL_ALERT_RECEIVED	Optional
NPF_NCI_SSL_MEM_FULL	Optional

Appendix A – npf_nci_ssl.h

```
/*
 * This header file defines typedefs, constants, and functions
 * for the NP Forum SSL API
 */
#ifndef __NPF_NCI_SSL_H
#define __NPF_NCI_SSL_H

#ifdef __cplusplus
extern "C"      {
#endif

/*-----------------------------------------------------------------
 *
 * Common Data Types
 *
 *-----------------------------------------------------------------*/
typedef NPF_uint32_t    NPF_NCI_SSLSessionNegHandle_t;
/*
 * This is a 32-bit value that is used to identify an SSL Session Negotiation
 * Database (SND).
 */

typedef NPF_uint32_t    NPF_NCI_SSLSessionHandle_t;
/*
 * This is a 32-bit value that is used to identify an SSL Session Database
 * (SSD).
 */

typedef NPF_uint32_t    NPF_NCI_SSLSessionCacheHandle_t;
/*
 * This is a 32-bit value that is used to identify an SSL Session Cache
 * Database (SSCD).
 */

typedef NPF_uint32_t    NPF_NCI_SSLTargetSelHandle_t;
/*
 * This is a 32-bit value that is used to identify an SSL Target Selector
 * Database (TSD).
 */

typedef NPF_uint32_t    NPF_NCI_SSLSessionId_t;
/*
 * This is a 32-bit value that is used to identify a client Session
 * (Session).
 */

typedef NPF_uint32_t    NPF_NCI_SSLSessionCacheId_t;
/*
 * This is a 32-bit value that is used to identify a cached client Session
 */

/*Constants that define the SSL Session attributes sizes*/
#define MAX_INITIAL_KEY_SIZE             256
#define MAX_SERVER_CERTIFICATE_SIZE      256
```

```
#define MAX_CLIENT_CA_LIST_SIZE          128

/*Constants that define the bulk data processing information sizes*/
#define MAX_SERVER_CIPHER_KEY_SIZE       4
#define MAX_CLIENT_CIPHER_KEY_SIZE       4
#define MAX_SERVER_MAC_KEY_SIZE          4
#define MAX_CLIENT_MAC_KEY_SIZE          4
#define MAX_SERVER_IV_KEY_SIZE           4
#define MAX_CLIENT_IV_KEY_SIZE           4

/*Size of a url inside the clear data, used to select the target server*/
#define MAX_URL_SIZE                     1024

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

/*
*     SSL Proxy Local address and port info
*/


typedef struct
{
      /* IP Version */
      /* IPv4 = 4, IPv6 = 6 */
      NPF_uint8_t       IP_Version;
      union {
            NPF_IPv4Address_t v4;              /* IPv4 address */
            NPF_IPv6Address_t v6;              /* IPv6 address */
            } addr;
      /* Transport Protocol */
      /* TCP =1 or SCTP = 2 */
      NPF_uint8_t       L4_Protocol;
      union {
         NPF_uint16_t   TCP_Port;
         NPF_uint32_t   SCTP_Port;
            } port;

} NPF_NCI_SSLAddressPortInfo_t;


/*
*     The NPF_NCI_SSLCipherAndDigest enum defines the various SSL ciphers and
      digests that can be supported by the SSL accelerator.
*/
typedef enum
{
      /* Standard Algorithm Identifiers */
```

```
      NPF_NCI_RSA_WITH_NULL_MD5 = 1,
      NPF_NCI_RSA_WITH_NULL_SHA,
      NPF_NCI_RSA_EXPORT_WITH_RC4_40_MD5,
      NPF_NCI_RSA_WITH_RC4_128_MD5,
      NPF_NCI_RSA_WITH_RC4_128_SHA,
      NPF_NCI_RSA_EXPORT_WITH_RC2_CBC_40_MD5,
      NPF_NCI_RSA_WITH_IDEA_CBC_SHA,
      NPF_NCI_RSA_EXPORT_WITH_DES40_CBC_SHA,
      NPF_NCI_RSA_WITH_DES_CBC_SHA,
      NPF_NCI_RSA_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_DH_DSS_EXPORT_WITH_DES40_CBC_SHA,
      NPF_NCI_DH_DSS_WITH_DES_CBC_SHA,
      NPF_NCI_DH_DSS_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_DH_RSA_EXPORT_WITH_DES40_CBC_SHA,
      NPF_NCI_DH_RSA_WITH_DES_CBC_SHA,
      NPF_NCI_DH_RSA_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA,
      NPF_NCI_DHE_DSS_WITH_DES_CBC_SHA,
      NPF_NCI_DHE_DSS_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA,
      NPF_NCI_DHE_RSA_WITH_DES_CBC_SHA,
      NPF_NCI_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_DH_anon_EXPORT_WITH_RC4_40_MD5,
      NPF_NCI_DH_anon_WITH_RC4_128_MD5,
      NPF_NCI_DH_anon_EXPORT_WITH_DES40_CBC_SHA,
      NPF_NCI_DH_anon_WITH_DES_CBC_SHA,
      NPF_NCI_DH_anon_WITH_3DES_EDE_CBC_SHA,
      NPF_NCI_RSA_EXPORT1024_WITH_DES_CBC_SHA,
      NPF_NCI_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA,
      NPF_NCI_RSA_EXPORT1024_WITH_RC4_56_SHA,
      NPF_NCI_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA,
      NPF_NCI_DHE_WITH_RC4_128_SHA,


      /* Experimental Algorithm Identifiers */

      NPF_NCI_START_OF_EXPERIMENTAL_SECTION=500,

      /* Proprietary Algorithm Identifiers */

      NPF_NCI_START_OF_PROPRIETARY_SECTION=1000
} NPF_NCI_SSLCipherAndDigest;


/*The following structure represents an entry in the SSL negotiation
Database*/

typedef struct {
   /* Address/Port Info */
   NPF_NCI_SSLAddressPortInfo_t     proxyAddress;

   /*cipher/digest supported*/
   NPF_NCI_SSLCipherAndDigest cipher_and_digest;

   /*Suggested Initial key*/
   NPF_uint8_t               initialKey[MAX_INITIAL_KEY_SIZE];
```

```
   /*Server's Public Certificate */
   NPF_uint8_t               certificate[MAX_SERVER_CERTIFICATE_SIZE];

   /*Server's Private Key */
   NPF_uint32_t              *privateKey;

   /*Comma separated, List of trusted CA's*/
   NPF_uint8_t               clientCAList[MAX_CLIENT_CA_LIST_SIZE];


} NPF_NCI_SSLSessionNegParams_t;

/* SSL negotiation Database array*/

typedef struct
{
      NPF_uint32_t nCount;
      NPF_NCI_SSLSessionNegParams_t* sndArray;
} NPF_NCI_SSLSessionNeg_Array_t;

/*Target selection policy type*/
typedef enum
{
      NPF_NCI_SSL_POLICY_PROXY_ADDR,      /*Indicates that the target is
                                          selected via the proxy address*/
      NPF_NCI_SSL_POLICY_URL              /*Indicates that the target is
                                          selected via the url in the data*/
} NPF_NCI_SSLPolicyType_t;

/*Target selection policy. An example of proxy external address and url in
the data is shown below*/

typedef struct {
   NPF_NCI_SSLPolicyType_t    policy_type;      /* policy type*/
   union
   {
      NPF_NCI_SSLIPAddress_t  proxyAddress;     /*External interface addr*/
      NPF_uint8_t             url[MAX_URL_SIZE];/*Url in clear data*/

      /*Other Proprietary fields go here */
   }u;
} NPF_NCI_SSLPolicy_t;

typedef struct {
   NPF_uint8_t               IP_Version;       /* IPv4 = 4, IPv6 = 6 */
   NPF_NCI_SSLIPAddress_t    targetAddress;    /* Target addr*/
   NPF_NCI_SSLPolicy_t       policy;
} NPF_NCI_SSLTargetSelectorParams_t;

/* SSL target selector Database array*/
typedef struct {
      NPF_uint32_t nCount;
      NPF_NCI_SSLTargetSelectorParams_t * tsdArray;
} NPF_NCI_SSLTargetSelector_Array_t;
```

```
/* SSL Connection information*/
typedef struct {
   NPF_uint32_t   clientConnID;     /*Identifies the client connection*/
   NPF_uint32_t   targetConnID;     /*Identifies the target connection*/
} NPF_NCI_SSLConnectionParams_t;

/* SSL Session Database*/
typedef struct {
   NPF_uint32_t               serverKeyInfo[MAX_SERVER_CIPHER_KEY_SIZE];
   NPF_uint32_t               clientKeyInfo[MAX_CLIENT_CIPHER_KEY_SIZE];
   NPF_uint32_t               serverMACKeyInfo[MAX_SERVER_MAC_KEY_SIZE];
   NPF_uint32_t               clientMACKeyInfo[MAX_CLIENT_MAC_KEY_SIZE];
   NPF_uint32_t               serverIVInfo[MAX_SERVER_IV_KEY_SIZE];
   NPF_uint32_t               clientIVInfo[MAX_CLIENT_IV_KEY_SIZE];
   NPF_NCI_SSLCipherAndDigest       cipherAndDigest;
   NPF_NCI_SSLConnectionParams_t    connectionMapping; /*SSL Client<-->target
                                                         mapping*/
} NPF_NCI_SSLSessionParams_t;

typedef struct {
      NPF_uint32_t                  nCount;
      NPF_NCI_SSLSessionParams_t    *sessionArray;
} NPF_NCI_SSLSession_Array_t;

typedef struct {
   NPF_uint32_t               serverKeyInfo[MAX_SERVER_CIPHER_KEY_SIZE];
   NPF_uint32_t               clientKeyInfo[MAX_CLIENT_CIPHER_KEY_SIZE];
   NPF_uint32_t               serverMACKeyInfo[MAX_SERVER_MAC_KEY_SIZE];
   NPF_uint32_t               clientMACKeyInfo[MAX_CLIENT_MAC_KEY_SIZE];
   NPF_uint32_t               serverIVInfo[MAX_SERVER_IV_KEY_SIZE];
   NPF_uint32_t               clientIVInfo[MAX_CLIENT_IV_KEY_SIZE];
   NPF_NCI_SSLCipherAndDigest       cipherAndDigest;
   NPF_NCI_SSLConnectionParams_t    connectionMapping; /*SSL Client<-->target
                                                         mapping*/
   NPF_uint32_t               sessionCacheTimeoutValue; /* in seconds */
} NPF_NCI_SSLSessionCacheParams_t;

typedef struct {
      NPF_uint32_t                  nCount;
      NPF_NCI_SSLSessionCacheParams_t     *sessionCacheArray;
} NPF_NCI_SSLSessionCache_Array_t;



/* SSL Data – encrypted/de-crypted */
typedef struct {
      NPF_uint32_t                  data_len;
      NPF_uint8_t                   *data;
} NPF_NCI_SSLData_t;

/*
      Crypto related stats
*/
typedef struct {
    NPF_uint32_t  HMAC_Errors;       /* HMAC (hash) errors -inbound only- */
    NPF_uint32_t  decryptErrors;     /* Decryption  errors -inbound*/
    NPF_uint32_t  encryptErrors;     /* Decryption  errors -outbound*/
```

```
    NPF_uint32_t  replayErrors;      /* Replay attacks -inbound only- */
  } NPF_NCI_SSLCryptoStats_t;

typedef struct {
      NPF_uint8_t             IP_Version;       /* IPv4 = 4, IPv6 = 6 */
      NPF_NCI_SSLIPAddress_t  clientAddr;
      NPF_NCI_SSLIPAddress_t  proxyAddr;
      NPF_NCI_SSLIPAddress_t  targetAddr;
}NPF_NCI_SSL_ConnectionInfo_t;

/*
      Connection related stats
*/
typedef struct {
  NPF_uint32_t                      numSessions;
  NPF_NCI_SSL_ConnectionInfo_t      *sslEndpoints;
} NPF_NCI_SSLConnectionStats_t;

/*
*     SSL reserved error codes in relation to other NPF APIs
*     Note** The maximum range is 100
*/
#define NPF_NCI_SSL_BASE_ERR 700 /* Base value of 700 wrt other NPF codes */

/*
*     SSL Error Type
*/
typedef NPF_uint32_t NPF_NCI_SSLErrorType_t;  /* SSL Error Type */

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

/* The Interface handle provided was not recognized as being valid */
#define NPF_NCI_SSL_E_INVALID_IF_HANDLE \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + 1)

/* Already registered */
#define NPF_NCI_SSL_E_ALREADY_REGISTERED \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + 2)

/* Optional feature not supported */
#define NPF_NCI_SSL_E_OPTIONAL_FEATURE_NOT_SUPPORTED \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + 3)

/* System was unable to allocate sufficient memory to complete operation */
#define NPF_NCI_SSL_E_NOMEMORY \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + 4)


/* Invalid SND handle */
#define NPF_NCI_SSL_E_INVALID_SND_HANDLE \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
```

```
      NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 1)

/* Invalid SSD handle */
#define NPF_NCI_SSL_E_INVALID_SSD_HANDLE \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
      NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 2)

/* Invalid TSD handle */
#define NPF_NCI_SSL_E_INVALID_TSD_HANDLE \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 3)

/* Invalid cipher algorithm */
#define NPF_NCI_SSL_E_INVALID_ENC_ALGO \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
      NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 4)

/* Invalid cipher algorithm keylen */
#define NPF_SSL_E_INVALID_ENC_ALGO_KEYLEN \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
      NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 5)

/* Invalid digest algorithm */
#define NPF_NCI_SSL_E_INVALID_DIGEST_ALGO \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
      NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 6)

/* Invalid digest algorithm keylen */
#define NPF_NCI_SSL_E_INVALID_DIGEST_ALGO_KEYLEN \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
       NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 7)

/* Invalid Initialization vector*/
#define NPF_NCI_SSL_E_INVALID_IV \
      ((NPF_NCI_SSLErrorType_t) NPF_NCI_SSL_BASE_ERR + \
       NPF_NCI_SSL_GENERIC_ERROR_CODE_COUNT + 8)


/*-----------------------------------------------------------------
 *
 * Completion Callback Data Types
 *
 *-----------------------------------------------------------------*/

/*Asynchronous Response*/

      typedef struct {
            NPF_NCI_SSLErrorType_t error;
            union {
                  NPF_NCI_SSLSessionId_t              sslSessionId;
                  NPF_NCI_SSLSessionCacheId_t         sslSessionCacheId;
                  NPF_NCI_SSLSessionNegHandle_t       sndHandle;
                  NPF_NCI_SSLTargetSelHandle_t        tsdHandle;
                  NPF_NCI_SSLSessionHandle_t          ssdHandle;
                  NPF_NCI_SSLSessionCacheHandle_t     sscdHandle;
                  NPF_NCI_SSLSessionNeg_Array_t       sndArray;
                  NPF_NCI_SSLTargetSelector_Array_t   tddArray;
```

```
                NPF_NCI_SSLSession_Array_t          ssdArray;
                NPF_NCI_SSLSessionCache_Array_t     sscdArray;
                NPF_NCI_SSLConnectionStats_t        sslConnStats;
                NPF_NCI_SSLCryptoStats_t            sslCryptoStats;
                NPF_NCI_SSLData_t                   ssl_data;
          } u;
     } NPF_NCI_SSLAsyncResponse_t;


/* completion callback types */
     typedef enum NPF_NCI_SSLCallbackType {
          NPF_NCI_SSL_SESSION_NEGOTIATION_DATABASE_CREATE  = 1,
          NPF_NCI_SSL_SESSION_NEGOTIATION_DATABASE_DELETE  = 2,
          NPF_NCI_SSL_SESSION_NEGOTIATION_INFO_ADD         = 3,
          NPF_NCI_SSL_SESSION_NEGOTIATION_INFO_DELETE      = 4,
          NPF_NCI_SSL_SESSION_NEGOTIATION_INFO_PURGE       = 5,
          NPF_NCI_SSL_SESSION_NEGOTIATION_DATABASE_QUERY   = 6,
          NPF_NCI_SSL_CONNECTION_STATISTICS_QUERY          = 7,
          NPF_NCI_SSL_CRYPTO_STATISTICS_QUERY              = 8,
          NPF_NCI_SSL_TARGET_SELECTOR_DATABASE_CREATE      = 9,
          NPF_NCI_SSL_TARGET_SELECTOR_DATABASE_DELETE      = 10,
          NPF_NCI_SSL_TARGET_SELECTOR_INFO_ADD             = 11,
          NPF_NCI_SSL_TARGET_SELECTOR_INFO_DELETE          = 12,
          NPF_NCI_SSL_TARGET_SELECTOR_INFO_PURGE           = 13,
          NPF_NCI_SSL_TARGET_SELECTOR_DATABASE_QUERY       = 14,
          NPF_NCI_SSL_SESSION_DATABASE_CREATE              = 15,
          NPF_NCI_SSL_SESSION_DATABASE_DELETE              = 16,
          NPF_NCI_SSL_SESSION_INFO_ADD                     = 17,
          NPF_NCI_SSL_SESSION_INFO_GET                     = 18,
          NPF_NCI_SSL_SESSION_INFO_DELETE                  = 19,
          NPF_NCI_SSL_SESSION_INFO_PURGE                   = 20,
          NPF_NCI_SSL_SESSION_DATABASE_QUERY               = 21,
          NPF_NCI_SSL_SESSION_CACHE_DATABASE_CREATE        = 22,
          NPF_NCI_SSL_SESSION_CACHE_DATABASE_DELETE        = 23,
          NPF_NCI_SSL_SESSION_CACHE_INFO_ADD               = 24,
          NPF_NCI_SSL_SESSION_CACHE_INFO_GET               = 25,
          NPF_NCI_SSL_SESSION_CACHE_INFO_DELETE            = 26,
          NPF_NCI_SSL_SESSION_CACHE_INFO_PURGE             = 27,
          NPF_NCI_SSL_SESSION_CACHE_DATABASE_QUERY         = 28,
          NPF_NCI_SSL_BULK_ENCRYPT_SEND                    = 29,
          NPF_NCI_SSL_BULK_ENCRYPT                         = 30,
          NPF_NCI_SSL_BULK_DECRYPT                         = 31
     } NPF_NCI_SSLCallbackType_t;

/*Callback Data*/

     typedef struct {
         NPF_NCI_SSLCallbackType_t     type;
         NPF_boolean_t                 allOK;
         NPF_uint32_t                  numResp;
         NPF_NCI_SSLAsyncResponse_t   *resp;
     } NPF_NCI_SSLCallbackData_t;

/*-------------------------------------------------------------
 *
 * Event Notification Data Types
 *
```

```
 *-------------------------------------------------------------------*/
/*
*     SSL Event
*/
typedef enum
{
      NPF_NCI_SSL_REPLAY_PACKET        = 1,/* Replay packet caught */
      NPF_NCI_SSL_DECRYPTION_FAILED    = 2,/* SSL decryption failed */
      NPF_NCI_SSL_ENCRYPTION_FAILED    = 3,/* SSL encryption failed */
      NPF_NCI_SSL_NO_TARGET_MAPPING    = 4,/* No target server found*/
      NPF_NCI_SSL_CLIENT_AUTH_FAILED   = 5,/* Client authentication failed*/
      NPF_NCI_SSL_ALERT_RECEIVED       = 6,/* Received an SSL Alert*/
      NPF_NCI_SSL_MEM_FULL             = 7 /*A crypto NPU ran out of memory*/
} NPF_NCI_SSLEvent_t;

#define SSL_MAX_EVENT_TYPES 7

/*
*     SSL event bitmask used in the event registration call.
*/
typedef NPF_uint32_t NPF_NCI_SSLEventMask_t;

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

#define NPF_NCI_SSL_EVENT_ALL_DISABLE                (0) /* disable all */
#define NPF_NCI_SSL_REPLAY_PACKET_ENABLE             (1 << 1)
#define NPF_NCI_SSL_DECRYPTION_FAILED_ENABLE         (1 << 2)
#define NPF_NCI_SSL_ENCRYPTION_FAILED_ENABLE         (1 << 3)
#define NPF_NCI_SSL_NO_TARGET_MAPPING_ENABLE         (1 << 4)
#define NPF_NCI_SSL_CLIENT_AUTH_FAILED_ENABLE        (1 << 5)
#define NPF_NCI_SSL_ALERT_RECEIVED_ENABLE            (1 << 6)
#define NPF_NCI_SSL_MEM_FULL_ENABLE                  (1 << 7)
#define NPF_SSL_EVENT_ALL_ENABLE                     0xFFFFFFFF


/*Problem packet*/
typedef struct
{
      NPF_uint8_t             IP_Version; /* 4 = IPv4, 6 = IPv6 */
      NPF_uint8_t             protocol;   /* IP Transport Protocol */
      NPF_NCI_SSLIPAddress_t  clientAddr; /*Address on client*/
      NPF_NCI_SSLIPAddress_t  proxyAddr;  /*Address on proxy interface*/
      NPF_uint16_t            srcPort;    /* IP Source Port */
      NPF_uint16_t            alertVal;   /* SSL alert value, 200 for others*/
} NPF_NCI_SSLPacketInfo_t;

/*
*     SSL Event Data
*/
typedef struct
{
      NPF_NCI_SSLEvent_t eventType;
      NPF_NCI_SSLPacketInfo_t packet;
} NPF_NCI_SSLEventData_t;
```

```
/*
*     SSL Event Array
*/
typedef struct
{
      NPF_uint16_t            n_data;           /*Number of events in array*/
      NPF_NCI_SSLEventData_t  *eventDataArray;  /* Array of event
                                                notifications */
} NPF_NCI_SSLEventArray_t;



/*-----------------------------------------------------------------
 *
 * Function Call Prototypes
 *
 *-----------------------------------------------------------------*/
typedef void (*NPF_NCI_SSLCallbackFunc_t)(
      NPF_IN NPF_userContext_t            userContext,
      NPF_IN NPF_correlator_t             correlator,
      NPF_IN NPF_NCI_SSLCallbackData_t    sslCallbackData);

typedef void (*NPF_NCI_SSLEventCallFunc_t) (
   NPF_IN NPF_userContext_t       userContext,
      NPF_IN NPF_NCI_SSLEventArray_t data);

NPF_error_t NPF_NCI_SSLRegister(
      NPF_IN NPF_userContext_t                  userContext,
      NPF_IN NPF_NCI_SSLCallbackFunc_t          sslCallbackFunc,
      NPF_OUT NPF_callbackHandle_t  *sslCallbackHandle);

      NPF_error_t NPF_NCI_SSLDeregister(
            NPF_IN NPF_callbackHandle_t   sslCallbackHandle);

      NPF_error_t NPF_NCI_SSLEventRegister(
         NPF_IN NPF_userContext_t               userContext,
         NPF_IN NPF_NCI_SSLEventCallFunc_t      eventCallFunc,
         NPF_IN NPF_NCI_SSLEventMask_t          eventMask,
         NPF_OUT NPF_callbackHandle_t           *eventCallHandle);

      NPF_error_t NPF_NCI_SSLEventDeregister(
         NPF_IN NPF_callbackHandle_t eventCallHandle);

      NPF_error_t    NPF_NCI_SSLSessionNegotiationDatabaseCreate(
                     NPF_IN   NPF_callbackHandle_t    cbHandle,
                     NPF_IN   NPF_correlator_t        correlator,
                     NPF_IN   NPF_errorReporting_t    cbDesired);

      NPF_error_t    NPF_NCI_SSLSessionNegotiationDatabaseDelete(
                     NPF_IN   NPF_callbackHandle_t          cbHandle,
                     NPF_IN   NPF_correlator_t              correlator,
                     NPF_IN   NPF_errorReporting_t          cbDesired,
                     NPF_IN   NPF_NCI_SSLSessionNegHandle_t sndHandle);

      NPF_error_t    NPF_NCI_SSLSessionNegotiationInfoAdd(
                        NPF_IN      NPF_callbackHandle_t    cbHandle,
```

```
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionNegHandle_t
                                                        sndHandle,
                    NPF_IN      NPF_uint32_t            numEntries,
                    NPF_IN      NPF_NCI_SSLSessionNegParams_t
                                                    *sessionArray);
NPF_error_t     NPF_NCI_SSLSeSessionNegotiationInfoDelete(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionNegHandle_t
                                                        sndHandle,
                    NPF_IN      NPF_uint32_t            numEntries,
                    NPF_IN      NPF_NCI_SSLSessionNegParams_t
                                                *sessionArray);
NPF_error_t     NPF_NCI_SSLSessionNegotiationInfoPurge(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionNegHandle_t
                                sndHandle);

NPF_error_t     NPF_NCI_SSLSessionNegotiationDatabaseQuery(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionNegHandle_t
                                                        sndHandle);

NPF_error_t     NPF_NCI_SSLSessionDatabaseCreate(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired);

NPF_error_t     NPF_NCI_SSLSessionDatabaseDelete(
                    NPF_IN   NPF_callbackHandle_t    cbHandle,
                    NPF_IN   NPF_correlator_t        correlator,
                    NPF_IN   NPF_errorReporting_t    cbDesired,
                    NPF_IN   NPF_NCI_SSLSessionHandle_t ssdHandle);

NPF_error_t     NPF_NCI_SSLSessionInfoAdd(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionHandle_t    ssdHandle,
                    NPF_IN      NPF_NCI_SSLSessionParams_t
                                                        *ssdEntry);
NPF_error_t     NPF_NCI_SSLSessionInfoGet(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionHandle_t
                                                        ssdHandle,
                    NPF_IN      NPF_NCI_SSLSessionParams_t
                                                        *ssdEntry,
                    NPF_OUT     NPF_NCI_SSLSessionParams_t
```

```
                                                    *ssdEntryOutput);
NPF_error_t    NPF_NCI_SSLSessionInfoDelete(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionHandle_t ssdHandle,
                  NPF_IN      NPF_uint32_t            numEntries,
                  NPF_IN      NPF_NCI_SSLSessionId_t
                                                    *ssdArray);
NPF_error_t    NPF_NCI_SSLSessionInfoPurge(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionHandle_t ssdHandle);

NPF_error_t    NPF_NCI_SSLSessionDatabaseQuery(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionHandle_t    ssdHandle);

NPF_error_t    NPF_NCI_SSLSessionCacheDatabaseCreate(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired);

NPF_error_t    NPF_NCI_SSLSessionCacheDatabaseDelete(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionCacheHandle_t
                                                    sscdHandle);

NPF_error_t    NPF_NCI_SSLSessionCacheInfoAdd(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionCacheHandle_t
                                                    sscdHandle,
                  NPF_IN      NPF_NCI_SSLSessionCacheParams_t
                                                    *sscdEntry);

NPF_error_t    NPF_NCI_SSLSessionCacheInfoGet(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
                  NPF_IN      NPF_NCI_SSLSessionCacheHandle_t
                                                    sscdHandle,
                  NPF_IN      NPF_NCI_SSLSessionCacheParams_t
                                                    *sscdEntry,
                  NPF_OUT     NPF_NCI_SSLSessionCacheParams_t
                                                 *sscdEntryOutput);
NPF_error_t    NPF_NCI_SSLSessionCacheInfoDelete(
                  NPF_IN      NPF_callbackHandle_t    cbHandle,
                  NPF_IN      NPF_correlator_t        correlator,
                  NPF_IN      NPF_errorReporting_t    cbDesired,
```

```
                    NPF_IN      NPF_NCI_SSLSessionCacheHandle_t
                                                    sscdHandle,
                    NPF_IN      NPF_uint32_t        numEntries,
                    NPF_IN      NPF_NCI_SSLSessionCacheId_t
                                                    *sscdArray);
NPF_error_t     NPF_NCI_SSLSessionCacheInfoPurge(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionCacheHandle_t
                                                    sscdHandle);
NPF_error_t     NPF_NCI_SSLSessionCacheDatabaseQuery(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionCacheHandle_t
                                                    sscdHandle);
NPF_error_t     NPF_NCI_SSLConnStatisticsQuery(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionNegHandle_t
                                                    sndHandle);
NPF_error_t     NPF_NCI_SSLCryptoStatisticsQuery (
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionNegHandle_t sndHandle);
NPF_error_t     NPF_NCI_SSLBulkEncryptSend(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionHandle_t    ssdHandle,
                    NPF_IN      NPF_uint8_t             *data,
                    NPF_IN      NPF_NCI_SSLSessionId_t  sslSession
                    );
NPF_error_t     NPF_NCI_SSLBulkEncrypt(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionHandle_t    ssdHandle,
                    NPF_IN      NPF_uint8_t             *data,
                    NPF_IN      NPF_NCI_SSLSessionId_t  sslSession
                    );
NPF_error_t     NPF_NCI_SSLBulkDecrypt(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired,
                    NPF_IN      NPF_NCI_SSLSessionHandle_t    ssdHandle,
                    NPF_IN      NPF_uint8_t             *data,
                    NPF_IN      NPF_NCI_SSLSessionId_t  sslSession
                    );
NPF_error_t     NPF_NCI_SSLTargetSelectorDatabaseCreate(
                    NPF_IN      NPF_callbackHandle_t    cbHandle,
                    NPF_IN      NPF_correlator_t        correlator,
                    NPF_IN      NPF_errorReporting_t    cbDesired
                    );
```

```
      NPF_error_t    NPF_NCI_SSLTargetSelectorDatabaseDelete(
                     NPF_IN   NPF_callbackHandle_t    cbHandle,
                     NPF_IN   NPF_correlator_t        correlator,
                     NPF_IN   NPF_errorReporting_t    cbDesired,
                     NPF_IN   NPF_NCI_SSLTargetSelHandle_t
                                                      tsdHandle);
      NPF_error_t    NPF_NCI_SSLTargetSelectorInfoAdd(
                        NPF_IN      NPF_callbackHandle_t    cbHandle,
                        NPF_IN      NPF_correlator_t        correlator,
                        NPF_IN      NPF_errorReporting_t    cbDesired,
                        NPF_IN      NPF_NCI_SSLTargetSelHandle_t
                                                            tsdHandle,
                        NPF_IN      NPF_uint32_t            numEntries,
                        NPF_IN      NPF_NCI_SSLTargetSelectorParams_t
                                                            *tsdArray);
      NPF_error_t    NPF_NCI_SSLTargetSelectorInfoDelete(
                        NPF_IN      NPF_callbackHandle_t    cbHandle,
                        NPF_IN      NPF_correlator_t        correlator,
                        NPF_IN      NPF_errorReporting_t    cbDesired,
                        NPF_IN      NPF_NCI_SSLTargetSelHandle_t
                                                            tsdHandle,
                        NPF_IN      NPF_uint32_t            numEntries,
                        NPF_IN      NPF_NCI_SSLTargetSelectorParams_t
                                                            *tsdArray);
      NPF_error_t    NPF_NCI_SSLTargetSelectorInfoPurge(
                        NPF_IN      NPF_callbackHandle_t    cbHandle,
                        NPF_IN      NPF_correlator_t        correlator,
                        NPF_IN      NPF_errorReporting_t    cbDesired,
                        NPF_IN      NPF_NCI_SSLTargetSelHandle_t
                                                            tsdHandle);
      NPF_error_t    NPF_NCI_SSLTargetSelectorDatabaseQuery(
                        NPF_IN      NPF_callbackHandle_t    cbHandle,
                        NPF_IN      NPF_correlator_t        correlator,
                        NPF_IN      NPF_errorReporting_t    cbDesired,
                        NPF_IN      NPF_NCI_SSLTargetSelHandle_t
                                                            tsdHandle);
#ifdef __cplusplus
}
#endif

#endif /* __NPF_NCI_SSL_H */
```

Appendix B Acknowledgements

Working Group Chair: Alex Conta, TranSwitch Corporation

Task Group Chair: Keith Williamson, Motorola Corporation

The following individuals are acknowledged for their participation in the Network Applications Task Group 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:

Suhail Ahmed, Intel

Asher Altman, Intel

Andy Gram, Nortel

Gary Hanson, RadiSys

Bernie Keany, Intel

David Lapp, Seaway

Keith Williamson, Motorola

Appendix C List of companies belonging to NPF During Approval Process

Agere Systems

AMCC

Analog Devices

Cypress Semiconductor

Enigma Semiconductor

Ericsson

Flextronics

Freescale Semiconductor

HCL Technologies

Hifn

IDT

Infineon Technologies AG

Intel

IP Fabrics

IP Infusion

Motorola

Mercury Computer Systems

Nokia

NTT Electronics

PMC Sierra

RadiSys

Sensory Networks

Sun Microsystems

Teja Technologies

TranSwitch

U4EA Group

Wintegra

Xelerated

Xilinx