Freescale Semiconductor
User’s Guide

SEC1SWUG
Rev. 0, 5/2006

Security Engine 1.0 Reference Device Driver Version 1.2

This user’s guide for the SEC1.0 reference device driver describes the security engine in several members of the PowerQUICC™ I and PowerQUICC II families of processors. The SEC 1.0 device driver specifically manages the operation of the following SEC cores:

- SEC 1.0, on the MPC8248, MPC8272 processors
- SEC 1.2, on the MPC875, MPC885 processors

The SEC 1.x reference driver supports all the crypto functionality of the SEC 1.0, which is a superset of the SEC 1.2. Consequently, not all the examples of application interaction with the SEC 1.x core apply to all PowerQUICC devices. Consult the individual PowerQUICC device reference manual to determine which SEC core and functions apply to your environment.

This document assumes familiarity with security engine 1.0 (SEC1) architecture and operation. A review of the SEC chapter of the *MPC8272 PowerQUICC™ II Family Reference Manual* or Part VIII of the *MPC885 PowerQUICC™ Family Reference Manual* is recommended.

Contents

1 SEC1 Basics

The SEC1 device driver is coded in ANSI C, and its operating system is designed to be as agnostic as practical. Where necessary, operating system dependencies are identified, and Section 11, “Porting the Driver,” addresses them. The SEC1 device driver was tested in VxWorks 5.5 and LinuxPPC using kernel versions 2.4.29, and 2.6.13.4.

Application interfaces to the SEC1 driver are implemented through the `ioctl()` function call. Requests made through this interface can be broken down into specific components, including miscellaneous requests and process requests. The miscellaneous requests are not related to the direct processing of data by the SEC1 core. Most SEC1 requests are process requests, and all execute using the same `ioctl()` access point. Section 4.6, “Process Request Structures,” describes the structures that compose these requests.

In this document, the crypto hardware accelerator (CHA) and execution unit (EU) acronyms are used interchangeably to refer to the device functional block that performs the crypto functions requested. For details on the device, consult the device reference manual. The design of this driver is a legacy holdover from two prior generations of security processors. Aspects of the interface for this driver are designed to maintain source-level application portability with previous versions of the driver/processor. Where relevant in this document, prior-version compatibility features are indicated.

2 Device Driver Components

Internally, the SEC1 device driver has four basic components (see Figure 1):

- Driver initialization and setup
- Application request processing
- Interrupt service routine
- Deferred service routine

When executing, the main driver code runs in the end-user application context, the interrupt service routine (ISR) runs in the interrupt context, and the processing complete indicator runs in its own context.

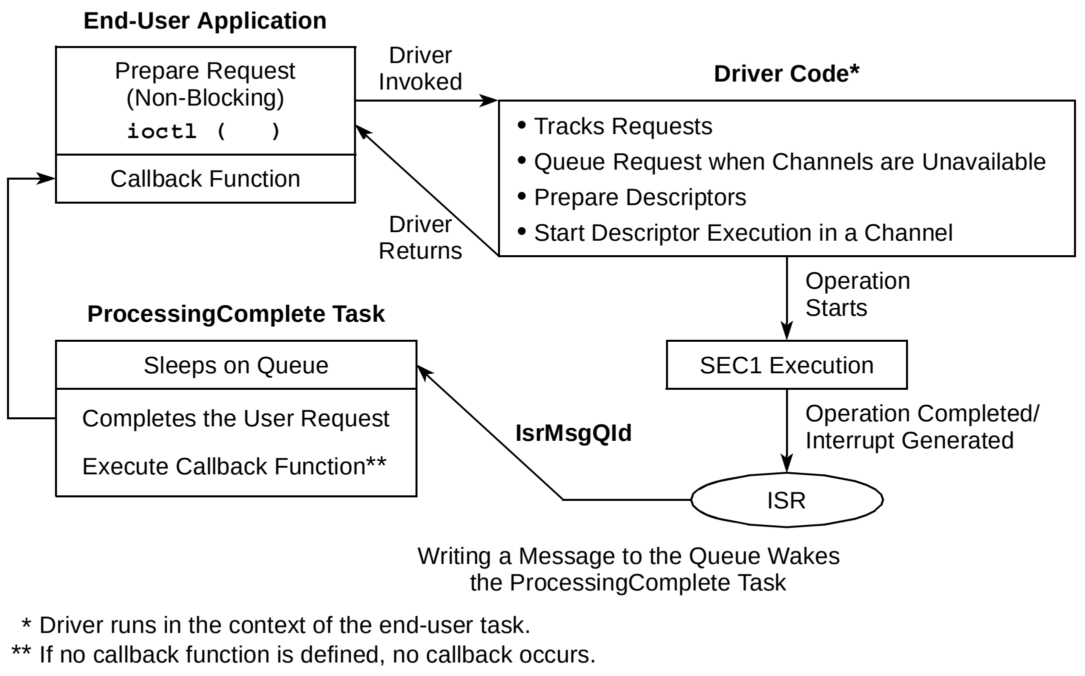
End-User Application
Prepare Request (Non-Blocking)
ioctl ( )
Callback Function
Driver Invoked
Driver Returns
Driver Code*
• Tracks Requests
• Queue Request when Channels are Unavailable
• Prepare Descriptors
• Start Descriptor Execution in a Channel
Operation Starts
SEC1 Execution
Operation Completed/ Interrupt Generated
ISR
IsrMsgQId
ProcessingComplete Task
Sleeps on Queue
Completes the User Request
Execute Callback Function**
Writing a Message to the Queue Wakes the ProcessingComplete Task
* Driver runs in the context of the end-user task.
** If no callback function is defined, no callback occurs.

Figure 1. Internal Driver Architecture

2.1 Driver Initialization Routine

The driver initialization routine includes both OS-specific and hardware-specific initialization. The driver initialization routine performs the following steps:

1. Finds the security engine core and sets the device memory map starting address in `IOBaseAddress`.
2. Initializes the SEC1 registers:
   — Controller registers
   — Channel registers
   — EU registers
3. Initializes the driver internal variables.
4. Initializes the channel assignments table.

   The device driver maintains this structure with state information for each channel and user request. A mutual exclusion semaphore protects this structure so multiple tasks are prevented from interfering with each other.
5. Initializes the internal request queue, which is a queue that holds requests to be dispatched when channels are available.

   The queue holds up to 16 requests. When the queue is full, the driver rejects requests with an error.
6. `ProcessingComplete()` is enabled and pends on `IsrMsgQId`, which serves as the interface between the interrupt service routine and this deferred task.

2.2 Request Dispatch Routine

The request dispatch routine provides the `ioctl()` interface to the device driver. It uses the caller request code to identify which function to execute and dispatches the appropriate routine to process the request. The driver tracks requests, queues requests when the requested channel is unavailable, prepares data packet descriptors, and writes the descriptor address to the appropriate channel, in effect giving the security engine the direction to begin processing the request. The `ioctl()` function returns to the end-user application without waiting for the security engine to complete, assuming that when hardware initiates a data packet descriptor (DPD) for processing, interrupt services can then invoke a handler to provide completion notification.

2.3 Process Request Routine

The process request routine translates the request into a sequence of one or more DPDs and feeds them to the security engine core to initiate processing. Dynamic requests are queued if no channels are available. For static channel requests, if the requested channel is not available or is busy, the requests fail and return an error.

NOTE

The SEC1.2 does not support static channel requests.

2.4 Interrupt Service Routine

When processing completes in the security engine, an interrupt is generated. The interrupt service routine handles the interrupt and queues the result of the operation in the `IsrMsgQId` queue to be processed by the `ProcessingComplete()` deferred service routine.

2.5 Deferred Service Routine

The `ProcessingComplete()` routine completes the request outside the interrupt service routine and runs in a non-ISR context. This task depends on the `IsrMsgQId` queue and processes messages written to the queue by the interrupt service routine. This function determines which request is complete and notifies the corresponding calling task using any handler specified by that calling task. It then checks the process request queue and schedules any queued requests.

3 User Interface

In order to make a request of the SEC1 device, the application must populate a request structure with the appropriate information to pass to the driver for processing. These structures are described in Section 4, "Global Definitions," and include such items as the operation ID, channel, callback routines (success and error), and data.

After the request is prepared, the end-user application calls `ioctl()` with the prepared request. This function is a standard system call used by operating system I/O subsystems to implement special-purpose functions. Its format is typically as follows:

```
int ioctl
  ( int fd,/* file descriptor */
    int function, /* function code */
    int arg/* arbitrary argument (driver dependent) */
  )
```

The function code (second argument) is defined as the I/O control code (see Section 4.1, “I/O Control Codes”). The third argument is the pointer to the SEC1 user request structure that contains the information needed by the driver to perform the function requested.

Following is a list of guidelines for the end-user application when a request structure is prepared:

- The first member of every request structure is an operation ID for use by the device driver to determine the format of the request structure.
- All process request structures have a channel member. For process requests that work in either dynamic or static mode, the channel can either be cleared to indicate dynamic mode or set to a valid (non-zero) channel number to indicate static mode. For process requests that work only in static mode, the channel should be set to a valid (non-zero) channel number.
- All process request structures have a `status` member. The device driver fills in this value when the interrupt for the operation occurs, and it reflects the status of the completed operation as determined by the deferred service routine.
- All process request structures have two notify members, `notify` and `notify_on_error`, for use by the SEC1 device driver to notify the application when its request is completed.
- All process request structures have a `next` request member so that an application can chain multiple process requests together.
- The application selects whether to use the callback function or to poll the status member to detect completion of the request.

3.1 Static Versus Dynamic Channels

Static mode allocates an execution unit (EU) to a specified channel. The EU does not need to reload internal registers every time. One advantage of static mode is faster internal EU execution. The disadvantages of static mode are the overhead of assigning and releasing the channel/EU and the fact that the channel and EU cannot be used by anyone else.

Static mode is appropriate when a single stream is processed—assign (loop on encrypt or decrypt) and release. Static mode is not appropriate when multiple interleaving streams are processed simultaneously—loop on (assign, encrypt or decrypt, release).

NOTE

SEC 1.2 does not support static channel requests.

The code to request and release a channel and EU is as follows:

```
.
      if (status = ioctl(device, IOCTL_RESERVE_CHANNEL_STATIC, &channel)) {
            printf("IOCTL_RESERVE_CHANNEL_STATIC failed 0x%04x\n", status);
            return status;
      }
      chan_st1 = channel;     /* save the channel number */
      channel = (channel << 8) | CHA_DES;
      if (status = ioctl(device, IOCTL_ASSIGN_CHA, &channel)) {
            printf("IOCTL_ASSIGN_CHA failed 0x%04x\n", status);
            return status;
      }
.
.
.
      if (status = ioctl(device, IOCTL_RELEASE_CHANNEL, &chan_st1))
            printf("IOCTL_RELEASE_CHANNEL failed 0x%04x\n", status);
```

3.2 Error Handling

The SEC1 device driver is asynchronous, so there are two main sources of errors:

- Syntax or logic. Errors are returned in the `status` member of the user request argument and as a return code from `ioctl()`. Errors of this type are detected by the driver, not by hardware.
- Protocol/procedure. These errors are returned only in the `status` member of the user request argument. Errors of this type are detected by hardware in the course of their execution.

Consequently, the end-user application needs two levels of error checking, the first one after the return from `ioctl()` and the second after the completion of the request. The second level is possible only if the request is made with a valid **`notify_on_error`** handler. If the handler is not specified, this level of error is lost. A code example of the two levels of errors follows, using an AES request:

```
AESA_CRYPT_REQ aesdynReq;
.
.
/* AES with dynamic aescriptor */
aesdynReq.opId = DPD_AESA_CBC_ENCRYPT_CRYPT;
aesdynReq.channel = 0;
aesdynReq.notify = (void *) notifAes;
aesdynReq.notify_on_error = (void *) notifAes;
aesdynReq.status = 0;
aesdynReq.inIvBytes = 16;
aesdynReq.inIvData = iv_in;
aesdynReq.keyBytes = 32;
aesdynReq.keyData = AesKey;
aesdynReq.inBytes = packet_length;
aesdynReq.inData = aesData;
aesdynReq.outData = aesResult;
aesdynReq.outIvBytes = 16;
aesdynReq.outIvData = iv_out;
aesdynReq.nextReq = 0;

status = Ioctl(device, IOCTL_PROC_REQ, &aesdynReq);
if (status != 0) {
printf ("Syntax-Logic Error in dynamic descriptor 0x%x\n", status);    .
  .
  .
}
.
/* in callback function notifAes    */
if (aesdynReq.status != 0) {
  printf ("Error detected by HW 0x%x\n", aesdynReq.status) ;
.
  .
 }
```

4 Global Definitions

Global definitions for the SEC1 device driver include I/O control codes, channel definitions, request operation ID masks, return codes, miscellaneous request structures, and process request structures.

4.1 I/O Control Codes

The I/O control code is the second argument to `ioctl()`. Internally, these values (as shown in Table 1) are used in conjunction with a base index to create the I/O control codes. The macro for this base index is defined by the **`SEC1_IOCTL_INDEX`** and has a default value of 0x0800.

Table 1. Second and Third Arguments in the `ioctl` Function

I/O Control Code (Second Argument in `ioctl` Function)	Third Argument in `ioctl` Function
IOCTL_PROC_REQ	Pointer to user's request structure.
IOCTL_GET_STATUS	Pointer to a STATUS_REQ.
IOCTL_RESERVE_CHANNEL_STATIC	Dedicate a static channel for use. Parameter is a pointer to the channel for reservation.
IOCTL_RESERVE_CHANNEL_MANUAL [1]	Manually dedicate a channel for debug. Parameter is a pointer to SEC1_RESERVE_MANUAL.
IOCTL_ASSIGN_CHA	Reserve a CHA for direct use. Parameter is an unsigned long identifying the CHA for reservation.
IOCTL_RELEASE_CHA	Free a CHA from direct use. Parameter is an unsigned long identifying the CHA to free.
IOCTL_RELEASE_CHANNEL	Free a channel reserved by IOCTL_RESERVE_CHANNEL_STATIC or IOCTL_RESERVE_CHANNEL_MANUAL
IOCTL_MALLOC	Allocate a contiguous block of kernel memory for processing a request. Parameter is a pointer to the allocated block. This argument is valid only on systems with privileged memory access.
IOCTL_FREE	Free a block of memory allocated by IOCTL_MALLOC. Parameter is a pointer to the block to free.
IOCTL_COPYFROM	Copy content from a user memory buffer to a kernel memory block allocated by IOCTL_MALLOC. Parameter is a pointer to a MALLOC_REQ.
IOCTL_COPYTO	Copy content from kernel memory block back to a user buffer. Parameter is a pointer to a MALLOC_REQ.

[1] This control code is used exclusively in debug/slave mode. The drivers make it available, but the function is not for general-purpose use.

4.2 Channel Definitions

The **NUM_CHANNELS** define specifies the number of channels in the security engine. For SEC1, the value is 4; for SEC 1.2, the value is 1.

Table 2. Channel Defines

Define	Description	Value For	
		SEC	SEC 1.2
NUM_AFHAS	Number of ARC4 CHAs	1	undefined [1]
NUM_DESAS	Number of DES CHAs	1	1
NUM_MDHAS	Number of MD CHAs	1	1
NUM_RNGAS	Number of RNG CHAs	1	undefined
NUM_PKHAS	Number of PK CHAs	1	undefined
NUM_AESAS	Number of AESA CHAs	1	1

[1] Undefined values for the SEC 1.2 reflect both the absence of that type of CHA and the absence of that #define.

The **NUM_CHAS** define contains the total number of crypto hardware accelerators (CHAs) in the SEC1 and in Table 2 is simply defined as the sum of the individual channels. The device used is defined as the macro **vxworksDrvName** regardless of whether VxWorks is used and regardless of whether the engine is SEC or SEC 1.2. It is set to `/dev/sec1` by default.

4.3 Request Operation ID (opId) Masks

Operation IDs can have two parts, the group or type of request and the request index or descriptor within a group or type (see Table 3). Operation IDs are provided to help programmers understand the structure of the opIds. They are not necessary in a user application.

Table 3. Request Operation ID Mask

Define	Description	Value
DESC_TYPE_MASK	The mask for the group or type of an `opId`.	0xFF00
DESC_NUM_MASK	The mask for the request index or descriptor within that group or type.	0x00FF

4.4 Return Codes

Table 4 provides a complete list of the error status results that may be returned to the callback routines.

Table 4. Callback Error Status Return Code

Define	Description	Value
SEC1_SUCCESS	Successful completion of request	0
SEC1__MEMORY_ALLOCATION	Driver cannot obtain memory from host OS	0xE004FFFF

Table 4. Callback Error Status Return Code (continued)

Define	Description	Value
`SEC1_INVALID_CHANNEL`	Channel specification out of range	0xE004FFFE
`SEC1_INVALID_CHA_TYPE`	Requested CHA does not exist	0xE004FFFD
`SEC1_INVALID_OPERATION_ID`	Requested `opID` is out of range for this request type	0xE004FFFC
`SEC1_CHANNEL_NOT_AVAILABLE`	Requested channel was not available.	0xE004FFFB
`SEC1_CHA_NOT_AVAILABLE`	Requested CHA not available when the request was processed.	0xE004FFFA
`SEC1_INVALID_LENGTH`	Length of requested data item is incompatible with request type, or data alignment is incompatible.	0xE004FFF9
`SEC1_OUTPUT_BUFFER_ALIGNMENT`	Output buffer alignment incompatible with request type.	0xE004FFF8
`SEC1_ADDRESS_PROBLEM`	Driver could not translate argued address into a physical address.	0xE004FFF6
`SEC1_INSUFFICIENT_REQS`	Request entry pool exhausted at the time of request processing; try again later.	0xE004FFF5
`SEC1_STATIC_CHANNEL_BUSY`	Requested static channel is already in use.	0xE004FFF3
`SEC1_CHA_ERROR`	CHA flagged an error during request processing; check the error notification context if provided in the request	0xE004FFF2
`SEC1_NULL_REQUEST`	Request pointer argued `NULL`.	0xE004FFF1
`SEC1_REQUEST_TIMED_OUT`	Timeout in request processing.	0xE004FFF0
`SEC1_MALLOC_FAILED`	Direct kernel memory buffer request failed.	0xE004FFEF
`SEC1_FREE_FAILED`	Direct kernel memory free failed.	0xE004FFEE
`SEC1_PARITY_SYSTEM_ERROR`	Parity Error detected on the bus.	0xE004FFED
`SEC1_INCOMPLETE_POINTER`	Error due to partial pointer.	0xE004FFEC
`SEC1_TEA_ERROR`	Transfer error.	0xE004FFEB
`SEC1_UNKNOWN_ERROR`	Any other unrecognized error.	0xE004FFEA
`SEC1_INVALID_DATA`	Error due to invalid request length.	0xE004FFE9
`SEC1_IO_MEMORY_ALLOCATE_ERROR`	Error due to insufficient resources.	–1001
`SEC1_IO_IO_ERROR`	Error due to I/O configuration.	–1002
`SEC1_IO_VXWORKS_DRIVER_TABLE_ADD_ERROR`	Error because VxWorks could not add driver to table.	–1003
`SEC1_IO_INTERRUPT_ALLOCATE_ERROR`	Error due to interrupt allocation error.	–1004
`SEC1_VXWORKS_CANNOT_CREATE_QUEUE`	Error because VxWorks could not create the ISR queue in IOInitQs().	–1009
`SEC1_CANCELLED_REQUEST`	Error due to canceled request.	–1010

Table 4. Callback Error Status Return Code (continued)

Define	Description	Value
`SEC1_INVALID_ADDRESS`	Error due to a NULL request.	–1011
`SEC1_TASK_ALREADY_INIT`	A task deferred for completion has already been started.	–1012

4.5 Miscellaneous Request Structures

The miscellaneous request structures discussed in this section are the STATUS_REQ structure and the SEC1_NOTIFY_ON_ERROR_CTX structure.

4.5.1 STATUS_REQ Structure

In the STATUS_REQ structure, each element is a copy of the contents of the same register in the SEC1 driver. This structure is also known as `SEC1_STATUS` through a `typedef`. The SEC version of the STATUS_REQ structure is as follows:

```
unsigned long ChaAssignmentStatusRegister[2];
unsigned long InterruptControlRegister[2];
unsigned long InterruptStatusRegister[2];
unsigned long IdRegister;
unsigned long ChannelStatusRegister[NUM_CHANNELS][2];
unsigned long ChannelConfigurationRegister[NUM_CHANNELS][2];
unsigned long CHAInterruptStatusRegister[NUM_CHAS][2];
unsigned long QueueEntryDepth;
unsigned long FreeChannels;
unsigned long FreeRngas;
unsigned long FreeAfhas;
unsigned long FreeDesas;
unsigned long FreeMdhas;
unsigned long FreePkhas;
unsigned long FreeAesas;
unsigned long FreeKeas;
unsigned long BlockSize;
```

The SEC 1.2 version of the STATUS_REQ structure is as follows:

```
unsigned long InterruptControlRegister[2];
unsigned long InterruptStatusRegister[2];
unsigned long IdRegister;
unsigned long ChannelStatusRegister[NUM_CHANNELS][2];
unsigned long ChannelConfigurationRegister[NUM_CHANNELS][2];
unsigned long CHAInterruptStatusRegister[NUM_CHAS][2];
unsigned long QueueEntryDepth;
unsigned long FreeChannels;
unsigned long FreeDesas;
unsigned long FreeMdhas;
unsigned long FreeAesas;
```

4.5.2 SEC1_NOTIFY_ON_ERROR_CTX Structure

The SEC1_NOTIFY_ON_ERROR_CTX structure is returned to the notify_on_error callback routine set up in the initial process request. This structure contains the original request structure as well as error and driver status.

```
unsigned long errorcode;    // Error that the request generated
void          *request;     // Pointer to original request
STATUS_REQ    driverstatus;// Details on the state of the hardware and the
                            //driver at the time of an error
```

- **`errorcode`**—error that the request generated
- **`*request`**—pointer to the original request
- **`driverstatus`**—details on the state of the hardware and the driver at the time of an error

4.6 Process Request Structures

All process request structures contain a copy of identical header information, which is defined by the `COMMON_REQ_PREAMBLE` structure, though no such structure is explicitly defined.

```
unsigned long                 opId;
unsigned char                 reserved;
unsigned char                 notifyFlags;
unsigned char                 reserved2;
unsigned char                 channel;
PSEC1_NOTIFY_ROUTINE          notify;
PSEC1_NOTIFY_CTX              pNotifyCtx;
PSEC1_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
SEC1_NOTIFY_ON_ERROR_CTX      ctxNotifyOnErr;
int                           status;
void                         *nextReq;
```

- `opId`–Operation ID to identify the type of the request. It is normally associated with a specific type of cryptographic operation.
- `notifyFlags`—If a POSIX-style signal handler handles request completion notification, it can contain ORed bits of `NOTIFY_IS_PID` and/or `NOTIFY_ERROR_IS_PID`, signifying that the `notify` or `notify_on_error` pointers are instead the process IDs (`getpid()`) of the task requesting a signal upon request completion.
- `channel`—Identifies a channel for the request.
- `notify`—Pointer to a notification callback routine to be called when the request successfully completes. It may instead be a process ID if a user-state signal handler flags completion. See `notifyFlags`.
- `pNotifyCtx`—Pointer to context area to be passed back through the notification routine.
- `notify_on_error`—Pointer to the notify on error routine to be called when the request unsuccessfully completes. It may instead be a process ID if a user-state signal handler flags completion. See `notifyFlags`.
- `ctxNotifyOnErr`—Context area that is filled in by the driver when there is an error.
- `status`—Contains the returned status of the request.

- `nextReq`—Pointer to the next request so that multiple requests can be linked together and sent through a single `ioctl` function call.

The additional data in the process request structures is specific to the request; refer to the specific structure.

5 Individual Request Types

The request types covered in this section are random number requests, DES process requests, ARC4 process requests, hash requests, HMAC requests, AES requests, integer public key requests, EEC public key requests, IPSec requests, and 802.11 protocol requests.

5.1 Random Number Requests

The following structure definition for the random number generation process request is not available in the SEC 1.2 driver.

5.1.1 RNG_REQ

```
COMMON_REQ_PREAMBLE
unsigned long rngBytes;
unsigned char* rngData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **`NUM_RNGA_DESC`** defines the number of descriptors in the **`DPD_RNG_GROUP`** that use this request. **`DPD_RNG_GROUP`** (0x1000) defines the group for all descriptors within this request.

Table 5. RNG_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
`DPD_RNG_GETRN`	0x1000	Generate a series of random values

5.2 DES Process Request Structures

The following sections provide structure definitions for DES process requests.

5.2.1 DES_LOADCTX_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long ivBytes;  /* 0 or 8 bytes */
unsigned char* ivData;
unsigned long keyBytes;  /* 8, 16, or 24 bytes */
unsigned char* keyData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **`NUM_DES_LOADCTX_STATIC_DESC`** defines the number of descriptors in the **`DPD_DES_SA_LDCTX_GROUP`** that use this request. **`DPD_DES_SA_LDCTX_GROUP`** (0x2000) defines the group for all descriptors in this request.

Table 6. DES_LOADCTX_STATIC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_SDES_CBC_ENCRYPT_SA_LDCTX`	0x2000	Load context from a static channel to encrypt in a single DES using CBC mode.
`DPD_SDES_CBC_DECRYPT_SA_LDCTX`	0x2001	Load context from a static channel to decrypt in a single DES using CBC mode.
`DPD_SDES_ECB_ENCRYPT_SA_LDCTX`	0x2002	Load context from a static channel to encrypt in a single DES using ECB mode.
`DPD_SDES_ECB_DECRYPT_SA_LDCTX`	0x2003	Load context from a static channel to decrypt in a single DES using ECB mode.
`DPD_TDES_CBC_ENCRYPT_SA_LDCTX`	0x2004	Load context from a static channel to encrypt in a triple DES using CBC mode.
`DPD_TDES_CBC_DECRYPT_SA_LDCTX`	0x2005	Load context from a static channel to decrypt in triple DES using CBC mode.
`DPD_TDES_ECB_ENCRYPT_SA_LDCTX`	0x2006	Load context from a static channel to encrypt in triple DES using ECB mode.
`DPD_TDES_ECB_DECRYPT_SA_LDCTX`	0x2007	Load context from a static channel to decrypt in triple DES using ECB mode.

5.2.2 DES_LOADCTX_CRYPT_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long ivBytes;  /* 0 or 8 bytes */
unsigned char* ivData;
unsigned long keyBytes;  /* 8, 16, or 24 bytes */
unsigned char* keyData;
unsigned long inBytes;  /* multiple of 8 bytes */
unsigned char* inData;
unsigned char* outData;  /* output length = input length */
unsigned long outCtxBytes;  /* 0 or 8 bytes */
unsigned char* outCtxData;/* MPCTEST: added outCtxData */
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **`NUM_DES_LOADCTX_CRYPT_STATIC_DESC`** defines the number of descriptors in the **`DPD_DES_SA_LDCTX_CRYPT_GROUP`** that use this request. **`DPD_DES_SA_LDCTX_CRYPT_GROUP`** (0x2100) defines the group for all descriptors in this request.

Table 7. DES_LOADCTX_CRYPT_STATIC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_SDES_CBC_ENCRYPT_SA_LDCTX_CRYPT`	0x2100	Load encrypted context from a static channel to encrypt in single DES using CBC mode.
`DPD_SDES_CBC_DECRYPT_SA_LDCTX_CRYPT`	0x2101	Load encrypted context from a static channel to decrypt in single DES using CBC mode.
`DPD_SDES_ECB_ENCRYPT_SA_LDCTX_CRYPT`	0x2102	Load encrypted context from a static channel to encrypt in single DES using ECB mode.

Table 7. DES_LOADCTX_CRYPT_STATIC_REQ Valid Descriptors (opId) (continued)

Descriptors	Value	Function Description
`DPD_SDES_ECB_DECRYPT_SA_LDCTX_CRYPT`	0x2103	Load encrypted context from a static channel to decrypt in single DES using ECB mode.
`DPD_TDES_CBC_ENCRYPT_SA_LDCTX_CRYPT`	0x2104	Load encrypted context from a static channel to encrypt in triple DES using CBC mode.
`DPD_TDES_CBC_DECRYPT_SA_LDCTX_CRYPT`	0x2105	Load encrypted context from a static channel to decrypt in triple DES using CBC mode.
`DPD_TDES_ECB_ENCRYPT_SA_LDCTX_CRYPT`	0x2106	Load encrypted context from a static channel to encrypt in triple DES using ECB mode.
`DPD_TDES_ECB_DECRYPT_SA_LDCTX_CRYPT`	0x2107	Load encrypted context from a static channel to decrypt in triple DES using ECB mode.

5.2.3 DES_CRYPT_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long inBytes;  /* multiple of 8 bytes */
unsigned char* inData;
unsigned char* outData;  /* output length = input length */
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **`NUM_DES_STATIC_DESC`** defines the number of descriptors in the **`DPD_DES_SA_CRYPT_GROUP`** that use this request. **`DPD_DES_SA_CRYPT_GROUP`** (0x2200) defines the group for all descriptors in this request.

Table 8. DES_CRYPT_STATIC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_SDES_CBC_ENCRYPT_SA_CRYPT`	0x2200	Encrypt data in a static channel in a single DES using CBC mode.
`DPD_SDES_CBC_DECRYPT_SA_CRYPT`	0x2201	Decrypt data in a static channel in a single DES using CBC mode.
`DPD_SDES_ECB_ENCRYPT_SA_CRYPT`	0x2202	Encrypt data in a static channel in a single DES using ECB mode.
`DPD_SDES_ECB_DECRYPT_SA_CRYPT`	0x2203	Decrypt data in a static channel in a single DES using ECB mode.
`DPD_TDES_CBC_ENCRYPT_SA_CRYPT`	0x2204	Encrypt data in a static channel in a triple DES using CBC mode.
`DPD_TDES_CBC_DECRYPT_SA_CRYPT`	0x2205	Decrypt data in a static channel in a triple DES using CBC mode.
`DPD_TDES_ECB_ENCRYPT_SA_CRYPT`	0x2206	Encrypt data in a static channel in a triple DES using ECB mode.
`DPD_TDES_ECB_DECRYPT_SA_CRYPT`	0x2207	Decrypt data in a static channel in a triple DES using ECB mode.

5.2.4 DES_CRYPT_GETCTX_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long inBytes;  /* multiple of 8 bytes */
unsigned char* inData;
unsigned char* outData;  /* output length = input length */
unsigned long outCtxBytes;  /* 0 or 8 bytes */
unsigned char* outCtxData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **`NUM_DES_CRYPT_UNLOADCTX_STATIC_DESC`** defines the number of descriptors in the **`DPD_DES_SA_CRYPT_ULCTX_GROUP`** that use this request. **`DPD_DES_SA_CRYPT_ULCTX_GROUP`** (0x2300) defines the group for all descriptors within this request.

Table 9. DES_CRYPT_GETCTX_STATIC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_SDES_CBC_ENCRYPT_SA_CRYPT_ULCTX`	0x2300	Get context from a static channel encrypted in a single DES using CBC mode.
`DPD_SDES_CBC_DECRYPT_SA_CRYPT_ULCTX`	0x2301	Get context from a static channel decrypted in single DES using CBC mode.
`DPD_SDES_ECB_ENCRYPT_SA_CRYPT_ULCTX`	0x2302	Get context from a static channel encrypted in single DES using ECB mode.
`DPD_SDES_ECB_DECRYPT_SA_CRYPT_ULCTX`	0x2303	Get context from a static channel decrypted in single DES using ECB mode.
`DPD_TDES_CBC_ENCRYPT_SA_CRYPT_ULCTX`	0x2304	Get context from a static channel encrypted in triple DES using CBC mode.
`DPD_TDES_CBC_DECRYPT_SA_CRYPT_ULCTX`	0x2305	Get context from a static channel decrypted in triple DES using CBC mode.
`DPD_TDES_ECB_ENCRYPT_SA_CRYPT_ULCTX`	0x2306	Get context from a static channel encrypted in triple DES using ECB mode.
`DPD_TDES_ECB_DECRYPT_SA_CRYPT_ULCTX`	0x2307	Get context from a static channel decrypted in triple DES using ECB mode.

5.2.5 DES_GETCTX_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long ivBytes;
unsigned char* ivData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **`NUM_DES_STATIC_ULCTX_DESC`** defines the number of descriptors in the **`DPD_DES_SA_ULCTX_GROUP`** that use this request. **`DPD_DES_SA_ULCTX_GROUP`** (0x2400) defines the group for all descriptors in this request.

Table 10. DES_GETCTX_STATIC_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
`DPD_DES_SA_ULCTX`	0x2400	Get context from a static channel that was encrypted single DES

5.2.6 DES_LOADCTX_CRYPT_REQ

```
COMMON_REQ_PREAMBLE
unsigned long inIvBytes;   /* 0 or 8 bytes */
unsigned char* inIvData;
unsigned long keyBytes;   /* 8, 16, or 24 bytes */
unsigned char* keyData;
```

```
    unsigned long inBytes;   /* multiple of 8 bytes */
    unsigned char* inData;
    unsigned char* outData;   /* output length = input length */
    unsigned long outIvBytes;   /* 0 or 8 bytes */
    unsigned char* outIvData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **`NUM_DES_LOADCTX_DESC`** defines the number of descriptors within the **`DPD_DES_CBC_CTX_GROUP`** that use this request. **`DPD_DES_CBC_CTX_GROUP`** (0x2500) defines the group for all descriptors within this request.

Table 11. DES_LOADCTX_CRYPT_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_SDES_CBC_CTX_ENCRYPT`	0x2500	Load encrypted context from a dynamic channel to encrypt in single DES using CBC mode.
`DPD_SDES_CBC_CTX_DECRYPT`	0x2501	Load encrypted context from a dynamic channel to decrypt in single DES using CBC mode.
`DPD_SDES_CBC_CTX_ENCRYPT`	0x2502	Load encrypted context from a dynamic channel to decrypt in single DES using CBC mode.
`DPD_SDES_CBC_CTX_DECRYPT`	0x2503	Load encrypted context from a dynamic channel to encrypt in single DES using CBC mode.

5.2.7 DES_CRYPT_REQ

```
    COMMON_REQ_PREAMBLE
    unsigned long keyBytes;   /* 8, 16, or 24 bytes */
    unsigned char* keyData;
    unsigned long inBytes;   /* multiple of 8 bytes */
    unsigned char* inData;
    unsigned char* outData;   /* output length = input length */
```

Dynamic channels are valid for this request. A channel value of zero is valid. **`NUM_DES_DESC`** defines the number of descriptors within the **`DPD_DES_ECB_GROUP`** that use this request. **`DPD_DES_ECB_GROUP`** (0x2600) defines the group for all descriptors within this request.

Table 12. DES_CRYPT_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_SDES_ECB_ENCRYPT`	0x2600	Load encrypted context from a dynamic channel to encrypt in single DES using ECB mode.
`DPD_SDES_ECB_DECRYPT`	0x2601	Load encrypted context from a dynamic channel to decrypt in single DES using ECB mode.
`DPD_TDES_ECB_ENCRYPT`	0x2602	Load encrypted context from a dynamic channel to decrypt in single DES using ECB mode.
`DPD_TDES_ECB_DECRYPT`	0x2603	Load encrypted context from a dynamic channel to encrypt in single DES using ECB mode.

5.3 ARC4 Process Request Structures

The following structure definitions for ARC4 process requests are not available in the SEC 1.2 driver.

5.3.1 ARC4_NEWCTX_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long keyBytes;
unsigned char* keyData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **NUM_RC4_STATIC_NEWCTX_DESC** defines the number of descriptors within the **DPD_RC4_SA_NEWCTX_GROUP** that use this request. **DPD_RC4_SA_NEWCTX_GROUP** (0x3000) defines the group for all descriptors within this request.

Table 13. ARC4_NEWCTX_STATIC_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
DPD_RC4_SA_NEWCTX	0x3000	Use RC4 on static channel with new context

5.3.2 ARC4_LOADCTX_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long ctxBytes;  /* 257 bytes */
unsigned char* ctxData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **NUM_RC4_STATIC_LOADCTX_DESC** defines the number of descriptors within the **DPD_RC4_SA_LDCTX_GROUP** that use this request. **DPD_RC4_SA_LDCTX_GROUP** (0x3100) defines the group for all descriptors within this request.

Table 14. ARC4_LOADCTX_STATIC_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
DPD_RC4_SA_LDCTX	0x3100	Load context from a static channel to encrypt using RC4.

5.3.3 ARC4_CRYPT_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long inBytes;
unsigned char* inData;
unsigned char* outData;  /* output length = input length */
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **NUM_RC4_STATIC_DESC** defines the number of descriptors within the DPD_RC4_SA_CRYPT_GROUP that use this request. **DPD_RC4_SA_CRYPT_GROUP** (0x3200) defines the group for all descriptors within this request.

Table 15. ARC4_CRYPTO_STATIC_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
`DPD_RC4_SA_CRYPT`	0x3200	Encrypt context from a static channel using RC4.

5.3.4 ARC4_CRYPT_GETCTX_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long inBytes;
unsigned char* inData;
unsigned char* outData;  /* output length = input length */
unsigned long outCtxBytes;  /* 257 bytes */
unsigned char* outCtxData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **`NUM_RC4_STATIC_UNLOADCTX_DESC`** defines the number of descriptors within the **`DPD_RC4_SA_CRYPT_ULCTX_GROUP`** that use this request. **`DPD_RC4_SA_CRYPT_ULCTX_GROUP`** (0x3300) defines the group for all descriptors within this request.

Table 16. ARC4_CRYPT_GETCTX_STATIC_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
`DPD_RC4_SA_CRYPT_ULCTX`	0x3300	Get context from a static channel that was encrypted using RC4

5.3.5 ARC4_LOADCTX_CRYPT_REQ

```
COMMON_REQ_PREAMBLE
unsigned long inCtxBytes;  /* 257 bytes */
unsigned char* inCtxData;
unsigned long inBytes;
unsigned char* inData;
unsigned char* outData;  /* output length = input length */
unsigned long outCtxBytes;  /* 257 bytes */
unsigned char* outCtxData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **`NUM_RC4_LOADCTX_UNLOADCTX_DESC`** defines the number of descriptors within the **`DPD_RC4_LDCTX_CRYPT_ULCTX_GROUP`** that use this request. **`DPD_RC4_LDCTX_CRYPT_ULCTX_GROUP`** (0x3400) defines the group for all descriptors within this request.

Table 17. ARC4_LOADCTX_CRYPT_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
`DPD_RC4_LDCTX_CRYPT_ULCTX`	0x3400	Load context from a dynamic channel to encrypt using RC4 and get the resulting context

5.3.6 ARC4_LOADKEY_CRYPT_UNLOADCTX_REQ

```
COMMON_REQ_PREAMBLE
unsigned long keyBytes;
unsigned char* keyData;
unsigned long inBytes;
unsigned char* inData;
unsigned char* outData;  /* output length = input length */
unsigned long outCtxBytes;  /* 257 bytes */
unsigned char* outCtxData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **NUM_RC4_LOADKEY_UNLOADCTX_DESC** defines the number of descriptors within the **DPD_RC4_LDKEY_CRYPT_ULCTX_GROUP** that use this request. **DPD_RC4_LDKEY_CRYPT_ULCTX_GROUP** (0x3500) defines the group for all descriptors within this request.

Table 18. ARC4_LOADKEY_CRYPT_UNLOADCTX_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
DPD_RC4_LDKEY_CRYPT_ULCTX	0x3500	Load the key to a dynamic channel to encrypt using RC4 and get the resulting context

5.4 Hash Request Structures

The following sections provide structure definitions for hash requests.

5.4.1 HASH_LOADCTX_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long ctxBytes;
unsigned char* ctxData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **NUM_MDHA_STATIC_LOADCTX_DESC** defines the number of descriptors within the **DPD_HASH_SA_LDCTX_GROUP** that use this request. **DPD_HASH_SA_LDCTX_GROUP** (0x4000) defines the group for all descriptors within this request.

Table 19. HASH_LOADCTX_STATIC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_SHA256_SA_LDCTX	0x4000	Load context into a static channel to use an SHA-256 hash algorithm.
DPD_MD5_SA_LDCTX	0x4001	Load context into a static channel to use an MD5 hash algorithm.
DPD_SHA_SA_LDCTX	0x4002	Load context into a static channel to use an SHA-1 hash algorithm.

5.4.2 HASH_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long inBytes;
unsigned char* inData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **NUM_MDHA_STATIC_DESC** defines the number of descriptors within the **DPD_HASH_SA_HASH_GROUP** that use this request. **DPD_HASH_SA_HASH_GROUP** (0x4100) defines the group for all descriptors in this request.

Table 20. HASH_STATIC_REQ Valid Descriptors (0x4100) (opId)

Descriptors	Value	Function Description
`DPD_SHA256_SA_HASH`	0x4100	Execute SHA-256 hash algorithm on the loaded context of a static channel.
`DPD_MD5_SA_HASH`	0x4101	Execute MD5 hash algorithm on the loaded context of a static channel.
`DPD_SHA_SA_HASH`	0x4102	Execute SHA-1 hash algorithm on the loaded context of a static channel.
`DPD_SHA256_SA_IDGS_HASH`	0x4103	Execute SHA-256 IDGS hash algorithm on the loaded context of a static channel.
`DPD_MD5_SA_IDGS_HASH`	0x4104	Execute MD5 IDGS hash algorithm on the loaded context of a static channel.
`DPD_SHA_SA_IDGS_HASH`	0x4105	Execute SHA-1 IDGS hash algorithm on the loaded context of a static channel.

`NUM_MDHA_STATIC_PAD_DESC` defines the number of descriptors within the **`DPD_HASH_SA_HASH_PAD_GROUP`** that use this request. **`DPD_HASH_SA_HASH_PAD_GROUP`** (0x4200) defines the group for all descriptors within this request.

Table 21. HASH_STATIC_REQ Valid Descriptors (0x4200) (opId)

Descriptors	Value	Function Description
`DPD_SHA256_SA_HASH_PAD`	0x4200	Execute SHA-256 hash algorithm on the loaded context of a static channel using padding.
`DPD_MD5_SA_HASH_PAD`	0x4201	Execute MD5 hash algorithm on the loaded context of a static channel using padding.
`DPD_SHA_SA_HASH_PAD`	0x4202	Execute SHA-1 hash algorithm on the loaded context of a static channel using padding.
`DPD_SHA256_SA_IDGS_HASH_PAD`	0x4203	Execute SHA-256 IDGS hash algorithm on the loaded context of a static channel using padding.
`DPD_MD5_SA_IDGS_HASH_PAD`	0x4204	Execute MD5 IDGS hash algorithm on the loaded context of a static channel using padding.
`DPD_SHA_SA_IDGS_HASH_PAD`	0x4205	Execute SHA-1 IDGS hash algorithm on the loaded context of a static channel using padding.

5.4.3 HASH_GETCTX_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long ctxBytes;
unsigned char* ctxData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **`NUM_MDHA_STATIC_UNLOAD_CTX_DESC`** defines the number of descriptors within the **`DPD_MD_SA_ULCTX_GROUP`** that use this request. **`DPD_MD_SA_ULCTX_GROUP`** (0x4300) defines the group for all descriptors within this request.

Table 22. HASH_GETCTX_STATIC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_SHA256_SA_ULCTX`	0x4300	Get context into a static channel that was used with an SHA-256 hash algorithm.
`DPD_MD5_SA_ULCTX`	0x4301	Get context into a static channel that was used with an MD5 hash algorithm.
`DPD_SHA_SA_ULCTX`	0x4302	Get context into a static channel that was used with an SHA-1 hash algorithm.

5.4.4 HASH_REQ

```
COMMON_REQ_PREAMBLE
unsigned long ctxBytes;
unsigned char* ctxData;
unsigned long inBytes;
unsigned char* inData;
unsigned long outBytes; /* length is fixed by algorithm */
unsigned char* outData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **`NUM_MDHA_DESC`** defines the number of descriptors within the **`DPD_HASH_LDCTX_HASH_ULCTX_GROUP`** that use this request. **`DPD_HASH_LDCTX_HASH_ULCTX_GROUP`** (0x4400) defines the group for all descriptors within this request.

Table 23. HASH_REQ Valid Descriptors (0x4400) (opId)

Descriptors	Value	Function Description
`DPD_SHA256_LDCTX_HASH_ULCTX`	0x4400	Load context into a dynamic channel to use an SHA-256 hash algorithm and get the resulting context.
`DPD_MD5_LDCTX_HASH_ULCTX`	0x4401	Load context into a dynamic channel to use an MD5 hash algorithm and get the resulting context.
`DPD_SHA_LDCTX_HASH_ULCTX`	0x4402	Load context into a dynamic channel to use an SHA-1 hash algorithm and get the resulting context.
`DPD_SHA256_LDCTX_IDGS_HASH_ULCTX`	0x4403	Load context into a dynamic channel to use an SHA-256 IDGS hash algorithm and get the resulting context.
`DPD_MD5_LDCTX_IDGS_HASH_ULCTX`	0x4404	Load context into a dynamic channel to use an MD5 IDGS hash algorithm and get the resulting context.
`DPD_SHA_LDCTX_IDGS_HASH_ULCTX`	0x4405	Load context into a dynamic channel to use an SHA-1 IDGS hash algorithm and get the resulting context.

`NUM_MDHA_PAD_DESC` defines the number of descriptors in the **`DPD_HASH_LDCTX_HASH_PAD_ULCTX_GROUP`** that use this request. **`DPD_HASH_LDCTX_HASH_PAD_ULCTX_GROUP`** (0x4500) defines the group for all descriptors in this request.

Table 24. HASH_REQ Valid Descriptors (0x4500) (opId)

Descriptors	Value	Function Description
`DPD_SHA256_LDCTX_HASH_PAD_ULCTX`	0x4500	Load context into a dynamic channel to use an SHA-256 hash algorithm and get the resulting padded context.
`DPD_MD5_LDCTX_HASH_PAD_ULCTX`	0x4501	Load context into a dynamic channel to use an MD5 hash algorithm and get the resulting padded context.
`DPD_SHA_LDCTX_HASH_PAD_ULCTX`	0x4502	Load context into a dynamic channel to use an SHA-1 hash algorithm and get the resulting padded context.
`DPD_SHA256_LDCTX_IDGS_HASH_PAD_ULCTX`	0x4503	Load context into a dynamic channel to use an SHA-256 IDGS hash algorithm and get the resulting padded context.
`DPD_MD5_LDCTX_IDGS_HASH_PAD_ULCTX`	0x4504	Load context into a dynamic channel to use an MD5 IDGS hash algorithm and get the resulting padded context.
`DPD_SHA_LDCTX_IDGS_HASH_PAD_ULCTX`	0x4505	Load context into a dynamic channel to use an SHA-1 IDGS hash algorithm and get the resulting padded context.

5.5 HMAC Request Structures

The following sections provide structure definitions for HMAC requests.

5.5.1 HMAC_PAD_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long keyBytes;
unsigned char* keyData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **NUM_HMAC_STATIC_PAD_DESC** defines the number of descriptors in the **DPD_HMAC_SA_PAD_GROUP** that use this request. **DPD_HMAC_SA_PAD_GROUP** (0x4600) defines the group for all descriptors in this request.

Table 25. HMAC_PAD_STATIC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_HMAC_SA_SHA256_PAD`	0x4600	Perform a HMAC operation on a static channel to use an SHA-256 hash algorithm with padding.
`DPD_HMAC_SA_MD5_PAD`	0x4601	Perform a HMAC operation on a static channel to use an MD5 hash algorithm with padding.
`DPD_HMAC_SA_SHA_PAD`	0x4602	Perform a HMAC operation on a static channel to use an SHA-1 hash algorithm with padding.
`DPD_HMAC_SA_SHA256_PAD_IDGS`	0x4603	Perform a HMAC operation on a static channel to use an SHA-256 hash algorithm with IDGS padding.
`DPD_HMAC_SA_MD5_PAD_IDGS`	0x4604	Perform a HMAC operation on a static channel to use an MD5 hash algorithm with IDGS padding.
`DPD_HMAC_SA_SHA_PAD_IDGS`	0x4605	Perform a HMAC operation on a static channel to use an SHA-1 hash algorithm with IDGS padding.

5.5.2 HMAC_PAD_HASH_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long keyBytes;
unsigned char* keyData;
unsigned long inBytes;
unsigned char* inData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **NUM_HMAC_STATIC_PAD_HASH_DESC** defines the number of descriptors in the **DPD_HMAC_SA_PAD_HASH_GROUP** that use this request. **DPD_HMAC_SA_PAD_HASH_GROUP** (0x4700) defines the group for all descriptors in this request.

Table 26. HMAC_PAD_HASH_STATIC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_HMAC_SA_SHA256_PAD_HASH	0x4700	Perform an HMAC operation on a static channel to use an SHA-256 hash algorithm with padding.
DPD_HMAC_SA_MD5_PAD_HASH	0x4701	Perform an HMAC operation on a static channel to use an MD5 hash algorithm with padding.
DPD_HMAC_SA_SHA_PAD_HASH	0x4702	Perform an HMAC operation on a static channel to use an SHA-1 hash algorithm with padding.
DPD_HMAC_SA_SHA256_PAD_IDGS_HASH	0x4703	Perform an HMAC operation on a static channel to use an SHA-256 hash algorithm with IDGS padding.
DPD_HMAC_SA_MD5_PAD_IDGS_HASH	0x4704	Perform an HMAC operation on a static channel to use an MD5 hash algorithm with IDGS padding.
DPD_HMAC_SA_SHA_PAD_IDGS_HASH	0x4705	Perform an HMAC operation on a static channel to use an SHA-1 hash algorithm with IDGS padding.

5.5.3 HMAC_PAD_REQ

```
COMMON_REQ_PREAMBLE
unsigned long keyBytes;
unsigned char* keyData;
unsigned long inBytes;
unsigned char* inData;
unsigned long outBytes;  /* length is fixed by algorithm */
unsigned char* outData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **NUM_HMAC_PAD_DESC** defines the number of descriptors in the **DPD_HASH_LDCTX_HMAC_ULCTX_GROUP** that use this request. **DPD_HASH_LDCTX_HMAC_ULCTX_GROUP** (0x4A00) defines the group for all descriptors in this request.

Table 27. HMAC_PAD_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_SHA256_LDCTX_HMAC_ULCTX`	0x4A00	Load context into a dynamic channel to use an SHA-256 hash algorithm and get the resulting HMAC context.
`DPD_MD5_LDCTX_HMAC_ULCTX`	0x4A01	Load context into a dynamic channel to use an MD5 hash algorithm and get the resulting HMAC context.
`DPD_SHA_LDCTX_HMAC_ULCTX`	0x4A02	Load context into a dynamic channel to use an SHA-1 hash algorithm and get the resulting HMAC context.
`DPD_SHA256_LDCTX_HMAC_PAD_ULCTX`	0x4A03	Load context into a dynamic channel to use an SHA-256 IDGS hash algorithm and get the resulting padded HMAC context.
`DPD_MD5_LDCTX_HMAC_PAD_ULCTX`	0x4A04	Load context into a dynamic channel to use an MD5 IDGS hash algorithm and get the resulting padded HMAC context.
`DPD_SHA_LDCTX_HMAC_PAD_ULCTX`	0x4A05	Load context into a dynamic channel to use an SHA-1 IDGS hash algorithm and get the resulting padded HMAC context.

5.6 AES Request Structures

This section provides structure definitions for AES requests.

5.6.1 AESA_CRYPT_REQ

```
COMMON_REQ_PREAMBLE
unsigned long keyBytes;  /* 16, 24, or 32 bytes */
unsigned char* keyData;
unsigned long inIvBytes;  /* 0 or 16 bytes */
unsigned char* inIvData;
unsigned long inBytes;  /* multiple of 8 bytes */
unsigned char* inData;
unsigned char* outData;  /* output length = input length */
unsigned long outCtxBytes;  /* 0 or 8 bytes */
unsigned char* outCtxData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **`NUM_AESA_CRYPT_DESC`** defines the number of descriptors in the **`DPD_AESA_CRYPT_GROUP`** that use this request. **`DPD_AESA_CRYPT_GROUP`** (0x6000) defines the group for all descriptors in this request.

Table 28. AESA_CRYPT_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_AESA_CBC_ENCRYPT_CRYPT`	0x6000	Encrypt in AESA using CBC mode.
`DPD_AESA_CBC_DECRYPT_CRYPT`	0x6001	Decrypt in AESA using CBC mode.
`DPD_AESA_CBC_DECRYPT_CRYPT_RDK`	0x6002	Decrypt in AESA using CBC mode with RDK.
`DPD_AESA_ECB_ENCRYPT_CRYPT`	0x6003	Encrypt in AESA using ECB mode.
`DPD_AESA_ECB_DECRYPT_CRYPT`	0x6004	Decrypt in AESA using ECB mode.

Table 28. AESA_CRYPT_REQ Valid Descriptors (opId) (continued)

Descriptors	Value	Function Description
DPD_AESA_ECB_DECRYPT_CRYPT_RDK	0x6005	Perform decryption in AESA using ECB mode with RDK
DPD_AESA_CTR_CRYPT	0x6006	Perform CTR in AESA

5.7 Integer Public Key Request Structures

The structure definitions for integer public key requests are not available in the SEC 1.2 driver.

5.7.1 MOD_EXP_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long aDataBytes;
unsigned char* aData;
unsigned long expBytes;
unsigned char* expData;
unsigned long outBytes;
unsigned char* outData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **NUM_MM_STATIC_EXP_DESC** defines the number of descriptors in the **DPD_MM_SA_EXP_GROUP** that use this request. **DPD_MM_SA_EXP_GROUP** (0x5000) defines the group for all descriptors in this request.

Table 29. MOD_EXP_STATIC_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
DPD_MM_SA_EXP	0x5000	Perform a MOD operation on the public key for a static channel.

5.7.2 MOD_EXP_REQ

```
COMMON_REQ_PREAMBLE
unsigned long aDataBytes;
unsigned char* aData;
unsigned long expBytes;
unsigned char* expData;
unsigned long modBytes;
unsigned char* modData;
unsigned long outBytes;
unsigned char* outData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **NUM_MM_EXP_DESC** defines the number of descriptors in the **DPD_MM_LDCTX_EXP_ULCTX_GROUP** that use this request. **DPD_MM_LDCTX_EXP_ULCTX_GROUP** (0x5100) defines the group for all descriptors in this request.

Table 30. MOD_EXP_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
DPD_MM_LDCTX_EXP_ULCTX	0x5100	Load context into a dynamic channel and return the resulting context from a MOD operation.

5.7.3 MOD_R2MODN_REQ

```
COMMON_REQ_PREAMBLE
unsigned long modBytes;
unsigned char* modData;
unsigned long outBytes;
unsigned char* outData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **NUM_MM_R2MODN_DESC** defines the number of descriptors in the **DPD_MM_LDCTX_R2MODN_ULCTX_GROUP** that use this request. **DPD_MM_LDCTX_R2MODN_ULCTX_GROUP** (0x5200) defines the group for all descriptors in this request.

Table 31. MOD_R2MODN_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
DPD_MM_LDCTX_R2MODN_ULCTX	0x5200	Perform an R2MOD operation on the public key for a static channel.

5.7.4 MOD_RRMODP_REQ

```
COMMON_REQ_PREAMBLE
unsigned long nBytes;
unsigned long pBytes;
unsigned char* pData;
unsigned long outBytes;
unsigned char* outData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **NUM_MM_RRMODP_DESC** defines the number of descriptors in the **DPD_MM_LDCTX_RRMODP_ULCTX_GROUP** that use this request. **DPD_MM_LDCTX_RRMODP_ULCTX_GROUP** (0x5300) defines the group for all descriptors in this request.

Table 32. MOD_RRMODP_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
DPD_MM_LDCTX_RRMODP_ULCTX	0x5300	Load context in a dynamic channel and return the resulting context from an RRMODP operation.

5.7.5 MOD_2OP_REQ

```
COMMON_REQ_PREAMBLE
unsigned long bDataBytes;
unsigned char* bData;
unsigned long aDataBytes;
unsigned char* aData;
unsigned long modBytes;
unsigned char* modData;
unsigned long outBytes;
unsigned char* outData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **NUM_MM_2OP_DESC** defines the number of descriptors in the **DPD_MM_LDCTX_2OP_ULCTX_GROUP** that use this request. **DPD_MM_LDCTX_2OP_ULCTX_GROUP** (0x5400) defines the group for all descriptors in this request.

Table 33. MOD_2OP_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_MM_LDCTX_MUL1_ULCTX`	0x5400	Load context into a dynamic channel and return the resulting context from a MUL1 operation.
`DPD_MM_LDCTX_MUL2_ULCTX`	0x5401	Load context into a dynamic channel and return the resulting context from a MUL2 operation.
`DPD_MM_LDCTX_ADD_ULCTX`	0x5402	Load context into a dynamic channel and return the resulting context from a ADD operation.
`DPD_MM_LDCTX_SUB_ULCTX`	0x5403	Load context into a dynamic channel and return the resulting context from a SUB operation.
`DPD_POLY_LDCTX_A0_B0_MUL1_ULCTX`	0x5404	Load context into a dynamic channel and return the resulting context from a A0-to-B0 MUL1 operation.
`DPD_POLY_LDCTX_A0_B0_MUL2_ULCTX`	0x5405	Load context into a dynamic channel and return the resulting context from an A0-to-B0 MUL2 operation.
`DPD_POLY_LDCTX_A0_B0_ADD_ULCTX`	0x5406	Load context into a dynamic channel and return the resulting context from an A0-to-B0 ADD operation.
`DPD_POLY_LDCTX_A1_B0_MUL1_ULCTX`	0x5407	Load context into a dynamic channel and return the resulting context from an A1-to-B0 MUL1 operation.
`DPD_POLY_LDCTX_A1_B0_MUL2_ULCTX`	0x5408	Load context into a dynamic channel and return the resulting context from an A1-to-B0 MUL2 operation.
`DPD_POLY_LDCTX_A1_B0_ADD_ULCTX`	0x5409	Load context into a dynamic channel and return the resulting context from an A1-to-B0 ADD operation.
`DPD_POLY_LDCTX_A2_B0_MUL1_ULCTX`	0x540A	Load context into a dynamic channel and return the resulting context from an A2-to-B0 MUL1 operation.
`DPD_POLY_LDCTX_A2_B0_MUL2_ULCTX`	0x540B	Load context into a dynamic channel and return the resulting context from an A2-to-B0 MUL2 operation.
`DPD_POLY_LDCTX_A2_B0_ADD_ULCTX`	0x540C	Load context into a dynamic channel and return the resulting context from an A2-to-B0 ADD operation.
`DPD_POLY_LDCTX_A3_B0_MUL1_ULCTX`	0x540D	Load context into a dynamic channel and return the resulting context from an A3-to-B0 MUL1 operation.
`DPD_POLY_LDCTX_A3_B0_MUL2_ULCTX`	0x540E	Load context into a dynamic channel and return the resulting context from an A3-to-B0 MUL2 operation.
`DPD_POLY_LDCTX_A3_B0_ADD_ULCTX`	0x540F	Load context into a dynamic channel and return the resulting context from an A3-to-B0 ADD operation.
`DPD_POLY_LDCTX_A0_B1_MUL1_ULCTX`	0x5410	Load context into a dynamic channel and return the resulting context from an A0-to-B1 MUL1 operation.
`DPD_POLY_LDCTX_A0_B1_MUL2_ULCTX`	0x5411	Load context into a dynamic channel and return the resulting context from an A-to-B MUL2 operation.
`DPD_POLY_LDCTX_A0_B1_ADD_ULCTX`	0x5412	Load context into a dynamic channel and return the resulting context from an A0-to-B1 ADD operation.
`DPD_POLY_LDCTX_A1_B1_MUL1_ULCTX`	0x5413	Load context into a dynamic channel and return the resulting context from an A1-to-B1 MUL1 operation.

Table 33. MOD_2OP_REQ Valid Descriptors (opId) (continued)

Descriptors	Value	Function Description
`DPD_POLY_LDCTX_A1_B1_MUL2_ULCTX`	0x5414	Load context into a dynamic channel and return the resulting context from an A1-to-B1 MUL2 operation.
`DPD_POLY_LDCTX_A1_B1_ADD_ULCTX`	0x5415	Load context into a dynamic channel and return the resulting context from an A1-to-B1 ADD operation.
`DPD_POLY_LDCTX_A2_B1_MUL1_ULCTX`	0x5416	Load context into a dynamic channel and return the resulting context from an A2-to-B1 MUL1 operation.
`DPD_POLY_LDCTX_A2_B1_MUL2_ULCTX`	0x5417	Load context into a dynamic channel and return the resulting context from an A2-to-B1 MUL2 operation.
`DPD_POLY_LDCTX_A2_B1_ADD_ULCTX`	0x5418	Load context into a dynamic channel and return the resulting context from an A2-to-B1 ADD operation.
`DPD_POLY_LDCTX_A3_B1_MUL1_ULCTX`	0x5419	Load context into a dynamic channel and return the resulting context from an A3-to-B1 MUL1 operation.
`DPD_POLY_LDCTX_A3_B1_MUL2_ULCTX`	0x541A	Load context into a dynamic channel and return the resulting context from an A3-to-B1 MUL2 operation.
`DPD_POLY_LDCTX_A3_B1_ADD_ULCTX`	0x541B	Load context into a dynamic channel and return the resulting context from an A3-to-B1 ADD operation.
`DPD_POLY_LDCTX_A0_B2_MUL1_ULCTX`	0x541C	Load context into a dynamic channel and return the resulting context from an A0-to-B2 MUL1 operation.
`DPD_POLY_LDCTX_A0_B2_MUL2_ULCTX`	0x541D	Load context into a dynamic channel and return the resulting context from an A0-to-B2 MUL2 operation.
`DPD_POLY_LDCTX_A0_B2_ADD_ULCTX`	0x541E	Load context into a dynamic channel and return the resulting context from an A0-to-B2ADD operation.
`DPD_POLY_LDCTX_A1_B2_MUL1_ULCTX`	0x541F	Load context into a dynamic channel and return the resulting context from an A1-to-B2 MUL1 operation.
`DPD_POLY_LDCTX_A1_B2_MUL2_ULCTX`	0x5420	Load context into a dynamic channel and return the resulting context from an A1-to-B2 MUL2 operation.
`DPD_POLY_LDCTX_A1_B2_ADD_ULCTX`	0x5421	Load context into a dynamic channel and return the resulting context from an A1-to-B2 ADD operation.
`DPD_POLY_LDCTX_A2_B2_MUL1_ULCTX`	0x5422	Load context into a dynamic channel and return the resulting context from an A2-to-B2 MUL1 operation.
`DPD_POLY_LDCTX_A2_B2_MUL2_ULCTX`	0x5423	Load context into a dynamic channel and return the resulting context from an A2-to-B2 MUL2 operation.
`DPD_POLY_LDCTX_A2_B2_ADD_ULCTX`	0x5424	Load context into a dynamic channel and return the resulting context from an A2-to-B2 ADD operation.
`DPD_POLY_LDCTX_A3_B2_MUL1_ULCTX`	0x5425	Load context into a dynamic channel and return the resulting context from an A3-to-B2 MUL1 operation.
`DPD_POLY_LDCTX_A3_B2_MUL2_ULCTX`	0x5426	Load context into a dynamic channel and return the resulting context from an A3-to-B2 MUL2 operation.
`DPD_POLY_LDCTX_A3_B2_ADD_ULCTX`	0x5427	Load context into a dynamic channel and return the resulting context from an A3-to-B2 ADD operation.

Table 33. MOD_2OP_REQ Valid Descriptors (opId) (continued)

Descriptors	Value	Function Description
`DPD_POLY_LDCTX_A0_B3_MUL1_ULCTX`	0x5428	Load context into a dynamic channel and return the resulting context from an A0-to-B3 MUL1 operation.
`DPD_POLY_LDCTX_A0_B3_MUL2_ULCTX`	0x5429	Load context into a dynamic channel and return the resulting context from an A0-to-B3 MUL2 operation.
`DPD_POLY_LDCTX_A0_B3_ADD_ULCTX`	0x542A	Load context into a dynamic channel and return the resulting context from an A0-to-B3 ADD operation.
`DPD_POLY_LDCTX_A1_B3_MUL1_ULCTX`	0x542B	Load context into a dynamic channel and return the resulting context from an A1-to-B3 MUL1 operation.
`DPD_POLY_LDCTX_A1_B3_MUL2_ULCTX`	0x542C	Load context into a dynamic channel and return the resulting context from an A1-to-B3 MUL2 operation.
`DPD_POLY_LDCTX_A1_B3_ADD_ULCTX`	0x542D	Load context into a dynamic channel and return the resulting context from an A1-to-B3 ADD operation.
`DPD_POLY_LDCTX_A2_B3_MUL1_ULCTX`	0x542E	Load context into a dynamic channel and return the resulting context from an A2-to-B3 MUL1 operation.
`DPD_POLY_LDCTX_A2_B3_MUL2_ULCTX`	0x542F	Load context into a dynamic channel and return the resulting context from an A2-to-B3 MUL2 operation.
`DPD_POLY_LDCTX_A2_B3_ADD_ULCTX`	0x5430	Load context into a dynamic channel and return the resulting context from an A2-to-B3 ADD operation.
`DPD_POLY_LDCTX_A3_B3_MUL1_ULCTX`	0x5431	Load context into a dynamic channel and return the resulting context from an A3-to-B3 MUL1 operation.
`DPD_POLY_LDCTX_A3_B3_MUL2_ULCTX`	0x5432	Load context into a dynamic channel and return the resulting context from an A3-to-B3 MUL2 operation.
`DPD_POLY_LDCTX_A3_B3_ADD_ULCTX`	0x5433	Load context into a dynamic channel and return the resulting context from an A3-to-B3 ADD operation.

5.7.6 MOD_CLR_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long aDataBytes;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **`NUM_MM_STATIC_CLR_DESC`** defines the number of descriptors in the **`DPD_MM_SA_CLR_GROUP`** that use this request. **`DPD_MM_SA_CLR_GROUP`** (0x5A00) defines the group for all descriptors in this request.

Table 34. MOD_CLR_STATIC_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
`DPD_MM_SA_CLR`	0x5A00	Clear the MOD context in a static channel.

5.8 ECC Public Key Request Structures

The structure definitions for ECC public key requests are not available in the SEC 1.2 driver.

5.8.1 ECC_LOADPOINTK_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long x1DataBytes;
unsigned char* x1Data;
unsigned long y1DataBytes;
unsigned char* y1Data;
unsigned long z1DataBytes;
unsigned char* z1Data;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **NUM_EC_STATIC_LOADCTX_DESC** defines the number of descriptors in the **DPD_EC_SA_LOADCTX_GROUP** that use this request. **DPD_EC_SA_LOADCTX_GROUP** (0x5500) defines the group for all descriptors in this request.

Table 35. ECC_LOADPOINTK_STATIC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_EC_SA_FP_AFF_LDCTX	0x5500	Load the context of a static channel for an electronic codebook for the FP AFF.
DPD_EC_SA_FP_PROJ_LDCTX	0x5501	Load the context of a static channel for an electronic codebook for the FP project.
DPD_EC_SA_F2M_AFF_LDCTX	0x5502	Load the context of a static channel for an electronic codebook for the F2M AFF.
DPD_EC_SA_F2M_PROJ_LDCTX	0x5503	Load the context of a static channel for an electronic codebook for the F2M project.

5.8.2 ECC_LOADPARAM_PMULT_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long aDataBytes;
unsigned char* aData;
unsigned long bDataBytes;
unsigned char* bData;
unsigned long r2DataBytes;
unsigned char* r2Data;
unsigned long kDataBytes;
unsigned char* kData;
unsigned long pDataBytes;
unsigned char* pData;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **NUM_EC_STATIC_kP_DESC** defines the number of descriptors in the **DPD_EC_SA_kP_GROUP** that use this request. **DPD_EC_SA_kP_GROUP** (0x5600) defines the group for all descriptors in this request.

Table 36. ECC_LOADPARAM_PMULT_STATIC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_EC_SA_FP_AFF_kP	0x5600	Load the context of a static channel for an electronic codebook for the FP AFF with a P multiplier.

Table 36. ECC_LOADPARAM_PMULT_STATIC_REQ Valid Descriptors (opId) (continued)

DPD_EC_SA_FP_PROJ_kP	0x5601	Load the context of a static channel for an electronic codebook for the FP project with a P multiplier.
DPD_EC_SA_F2M_AFF_kP	0x5602	Load the context of a static channel for an electronic codebook for the F2M AFF with a P multiplier.
DPD_EC_SA_F2M_PROJ_kP	0x5603	Load the context of a static channel for an electronic codebook for the F2M project with a P multiplier.

5.8.3 ECC_GETRESULT_STATIC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long x2DataBytes;
unsigned char* x2Data;
unsigned long y2DataBytes;
unsigned char* y2Data;
unsigned long z2DataBytes;
unsigned char* z2Data;
unsigned long z_2DataBytes;
unsigned char* z_2Data;
unsigned long z_3DataBytes;
unsigned char* z_3Data;
```

Dynamic channels are not valid for this request. A channel value of zero is invalid. **NUM_EC_STATIC_UNLOAD_CTX_DESC** defines the number of descriptors in the **DPD_EC_SA_ULCTX_GROUP** that use this request. **DPD_EC_SA_ULCTX_GROUP** (0x5700) defines the group for all descriptors in this request.

Table 37. ECC_GETRESULT_STATIC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_EC_SA_FP_AFF_ULCTX	0x5700	Get the context from a static channel for an electronic codebook for the FP AFF.
DPD_EC_SA_FP_PROJ_ULCTX	0x5701	Get the context from a static channel for an electronic codebook for the FP project.
DPD_EC_SA_F2M_AFF_ULCTX	0x5702	Get the context from a static channel for an electronic codebook for the F2M AFF.
DPD_EC_SA_F2M_PROJ_ULCTX	0x5703	Get the context from a static channel for an electronic codebook for the F2M project.

5.8.4 ECC_POINT_REQ

```
COMMON_REQ_PREAMBLE
unsigned long par2DataBytes;
unsigned char* par2Data;
unsigned long par1DataBytes;
unsigned char* par1Data;
unsigned long expDataBytes;
unsigned char* expData;
unsigned long pDataBytes;
unsigned char* pData;
unsigned long pOutDataBytes;
unsigned char* pOutData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **`NUM_EC_POINT_DESC`** defines the number of descriptors in the **`DPD_EC_LDCTX_kP_ULCTX_GROUP`** that use this request. **`DPD_EC_LDCTX_kP_ULCTX_GROUP`** (0x5800) defines the group for all descriptors in this request.

Table 38. ECC_POINT_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_EC_FP_AFF_LDCTX_kP_ULCTX`	0x5800	Load context into a dynamic channel to use an electronic codebook for the FP AFF and get the resulting P multiplier context
`DPD_EC_FP_PROJ_LDCTX_kP_ULCTX`	0x5801	Load context into a dynamic channel to use an electronic codebook for the FP project and get the resulting P multiplier context
`DPD_EC_F2M_AFF_LDCTX_kP_ULCT`	0x5802	Load context into a dynamic channel to use an electronic codebook for the F2M AFF and get the resulting P multiplier context
`DPD_EC_F2M_PROJ_LDCTX_kP_ULCTX`	0x5803	Load context into a dynamic channel to use an electronic codebook for the F2M project and get the resulting P multiplier context
`DPD_EC_FP_LDCTX_ADD_ULCT`	0x5804	Load context into a dynamic channel to use an electronic codebook for the FP and get the resulting context from an add operation
`DPD_EC_FP_LDCTX_DOUBLE_ULCTX`	0x5805	Load context into a dynamic channel to use an electronic codebook for the FP and get the resulting context from a double operation
`DPD_EC_F2M_LDCTX_ADD_ULCTX`	0x5806	Load context into a dynamic channel to use an electronic codebook for the F2M and get the resulting context from an add operation
`DPD_EC_F2M_LDCTX_DOUBLE_ULCTX`	0x5807	Load context into a dynamic channel to use an electronic codebook for the F2M and get the resulting context from a double operation

5.8.5 ECC_2OP_REQ

```
COMMON_REQ_PREAMBLE
unsigned long bDataBytes;
unsigned char* bData;
unsigned long aDataBytes;
unsigned char* aData;
unsigned long modBytes;
unsigned char* modData;
unsigned long outBytes;
unsigned char* outData;
```

Dynamic channels are valid for this request. A channel value of zero is valid. **NUM_EC_2OP_DESC** defines the number of descriptors within the **DPD_EC_2OP_GROUP** that use this request. **DPD_EC_2OP_GROUP** (0x5900) defines the group for all descriptors within this request.

Table 39. ECC_2OP_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
DPD_EC_F2M_LDCTX_MUL1_ULCTX	0x5900	Load context into a dynamic channel to use an electronic codebook for the F2M and get the resulting context from a MULT1 operation

5.9 IPSec Request Structures

The following sections provide structure definitions for IPSec requests.

5.9.1 IPSEC_CBC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long hashKeyBytes;
unsigned char* hashKeyData;
unsigned long cryptKeyBytes;
unsigned char* cryptKeyData;
unsigned long cryptCtxInBytes;
unsigned char* cryptCtxInData;
unsigned long hashInDataBytes;
unsigned char* hashInData;
unsigned long inDataBytes;
unsigned char* inData;
unsigned char* cryptDataOut;
unsigned long hashDataOutBytes;
unsigned char* hashDataOut;
```

Dynamic and static channels are valid for this request. **NUM_IPSEC_CBC_DESC** defines the number of descriptors in the **DPD_IPSEC_CBC_GROUP** that use this request. **DPD_IPSEC_CBC_GROUP** (0x7000) defines the group for all descriptors in this request.

Table 40. IPSec_CBC_REQ Valid Descriptors (opId) for Dynamic Requests

Descriptors	Value	Function Description
DPD_IPSEC_CBC_SDES_ENCRYPT_MD5_PAD	0x7000	Perform IPSec encryption in single DES using CBC mode with MD5 padding.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA_PAD	0x7001	Perform IPSec encryption in single DES using CBC mode with SHA-1 padding.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA256_PAD	0x7002	Perform IPSec encryption in single DES using CBC mode with SHA-256 padding.
DPD_IPSEC_CBC_SDES_DECRYPT_MD5_PAD	0x7003	Perform IPSec decryption in single DES using CBC mode with MD5 padding.
DPD_IPSEC_CBC_SDES_DECRYPT_SHA_PAD	0x7004	Perform IPSec decryption in single DES using CBC mode with SHA-1 padding.

Table 40. IPSec_CBC_REQ Valid Descriptors (opId) for Dynamic Requests (continued)

Descriptors	Value	Function Description
DPD_IPSEC_CBC_SDES_DECRYPT_SHA256_PAD	0x7005	Perform IPSec decryption in single DES using CBC mode with SHA-256 padding.
DPD_IPSEC_CBC_TDES_ENCRYPT_MD5_PAD	0x7006	Perform IPSec encryption in triple DES using CBC mode with MD5 padding.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA_PAD	0x7007	Perform IPSec encryption in triple DES using CBC mode with SHA-1 padding.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA256_PAD	0x7008	Perform IPSec encryption in triple DES using CBC mode with SHA-256 padding.
DPD_IPSEC_CBC_TDES_DECRYPT_MD5_PAD	0x7009	Perform IPSec decryption in triple DES using CBC mode with MD5 padding.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA_PAD	0x700A	Perform IPSec decryption in triple DES using CBC mode with SHA-1 padding.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA256_PAD	0x700B	Perform IPSec decryption in triple DES using CBC mode with SHA-256 padding.
DPD_IPSEC_CBC_SDES_ENCRYPT_MD5	0x700C	Perform IPSec encryption in single DES using CBC mode with MD5.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA	0x700D	Perform IPSec encryption in single DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA256	0x700E	Perform IPSec encryption in single DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_SDES_DECRYPT_MD5	0x700F	Perform IPSec decryption in single DES using CBC mode with MD5.
DPD_IPSEC_CBC_SDES_DECRYPT_SHA	0x7010	Perform IPSec decryption in single DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_SDES_DECRYPT_SHA256	0x7011	Perform IPSec decryption in single DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_TDES_ENCRYPT_MD5	0x7012	Perform IPSec encryption in triple DES using CBC mode with MD5.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA	0x7013	Perform IPSec encryption in triple DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA256	0x7014	Perform IPSec encryption in triple DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_TDES_DECRYPT_MD5	0x7015	Perform IPSec decryption in triple DES using CBC mode with MD5.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA	0x7016	Perform IPSec decryption in triple DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA256	0x7017	Perform IPSec decryption in triple DES using CBC mode with SHA-256.

NUM_IPSEC_STATIC_CBC_DESC defines the number of descriptors in the DPD_IPSEC_STATIC_CBC_GROUP that use this request. DPD_IPSEC_STATIC_CBC_GROUP (0x7A00) defines the group for all descriptors in this request.

Table 41. IPSec_CBC_REQ Valid Descriptors (opId) for Static Requests

Descriptors	Value	Function Description
DPD_IPSEC_CBC_SDES_ENCRYPT_MD5_INIT	0x7A00	Perform the IPSec initialization for encrypting in single DES using CBC mode with MD5.
DPD_IPSEC_CBC_SDES_ENCRYPT_MD5_UPDATE	0x7A01	Perform the IPSec update for encrypting in single DES using CBC mode with MD5.
DPD_IPSEC_CBC_SDES_ENCRYPT_MD5_APAD_FINAL	0x7A02	Perform the IPSec APAD finalization for encrypting in single DES using CBC mode with MD5.
DPD_IPSEC_CBC_SDES_ENCRYPT_MD5_FINAL	0x7A03	Perform the IPSec finalization for encrypting in single DES using CBC mode with MD5.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA_INIT	0x7A04	Perform the IPSec initialization for encrypting in single DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA_UPDATE	0x7A05	Perform the IPSec update for encrypting in single DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA_APAD_FINAL	0x7A06	Perform the IPSec APAD finalization for encrypting in single DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA_FINAL	0x7A07	Perform the IPSec finalization for encrypting in single DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA256_INIT	0x7A08	Perform the IPSec initialization for encrypting in single DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA256_UPDATE	0x7A09	Perform the IPSec update for encrypting in single DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA256_APAD_FINAL	0x7A0A	Perform the IPSec APAD finalization for encrypting in single DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA256_FINAL	0x7A0B	Perform the IPSec finalization for encrypting in single DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_SDES_DECRYPT_MD5_INIT	0x7A0C	Perform the IPSec initialization for decrypting in single DES using CBC mode with MD5.
DPD_IPSEC_CBC_SDES_DECRYPT_MD5_UPDATE	0x7A0D	Perform the IPSec update for decrypting in single DES using CBC mode with MD5.
DPD_IPSEC_CBC_SDES_DECRYPT_MD5_APAD_FINAL	0x7A0E	Perform the IPSec APAD finalization for decrypting in single DES using CBC mode with MD5.
DPD_IPSEC_CBC_SDES_DECRYPT_MD5_FINAL	0x7A0F	Perform the IPSec finalization for decrypting in single DES using CBC mode with MD5.
DPD_IPSEC_CBC_SDES_DECRYPT_SHA_INIT	0x7A10	Perform the IPSec initialization for decrypting in single DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_SDES_DECRYPT_SHA_UPDATE	0x7A11	Perform the IPSec update for decrypting in single DES using CBC mode with SHA-1.

Table 41. IPSec_CBC_REQ Valid Descriptors (opId) for Static Requests (continued)

Descriptors	Value	Function Description
DPD_IPSEC_CBC_SDES_DECRYPT_SHA_APAD_FINAL	0x7A12	Perform the IPSec APAD finalization for decrypting in single DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_SDES_DECRYPT_SHA_FINAL	0x7A13	Perform the IPSec finalization for decrypting in single DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_SDES_DECRYPT_SHA256_INIT	0x7A14	Perform the IPSec initialization for decrypting in single DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_SDES_DECRYPT_SHA256_UPDATE	0x7A15	Perform the IPSec update for decrypting in single DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_SDES_DECRYPT_SHA256_APAD_FINAL	0x7A16	Perform the IPSec APAD finalization for decrypting in single DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_SDES_DECRYPT_SHA256_FINAL	0x7A17	Perform the IPSec finalization for decrypting in single DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_TDES_ENCRYPT_MD5_INIT	0x7A18	Perform the IPSec initialization for encrypting in triple DES using CBC mode with MD5.
DPD_IPSEC_CBC_TDES_ENCRYPT_MD5_UPDATE	0x7A19	Perform the IPSec update for encrypting in triple DES using CBC mode with MD5.
DPD_IPSEC_CBC_TDES_ENCRYPT_MD5_APAD_FINAL	0x7A1A	Perform the IPSec APAD finalization for encrypting in triple DES using CBC mode with MD5.
DPD_IPSEC_CBC_TDES_ENCRYPT_MD5_FINAL	0x7A1B	Perform the IPSec finalization for encrypting in triple DES using CBC mode with MD5.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA_INIT	0x7A1C	Perform the IPSec initialization for encrypting in triple DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA_UPDATE	0x7A1D	Perform the IPSec update for encrypting in triple DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA_APAD_FINAL	0x7A1E	Perform the IPSec APAD finalization for encrypting in triple DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA_FINAL	0x7A1F	Perform the IPSec finalization for encrypting in triple DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA256_INIT	0x7A20	Perform the IPSec initialization for encrypting in triple DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA256_UPDATE	0x7A21	Perform the IPSec update for encrypting in triple DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA256_APAD_FINAL	0x7A22	Perform the IPSec APAD finalization for encrypting in triple DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_TDES_ENCRYPT_SHA256_FINAL	0x7A23	Perform the IPSec finalization for encrypting in triple DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_TDES_DECRYPT_MD5_INIT	0x7A24	Perform the IPSec initialization for decrypting in triple DES using CBC mode with MD5.
DPD_IPSEC_CBC_TDES_DECRYPT_MD5_UPDATE	0x7A25	Perform the IPSec update for decrypting in triple DES using CBC mode with MD5.

Table 41. IPSec_CBC_REQ Valid Descriptors (opId) for Static Requests (continued)

Descriptors	Value	Function Description
DPD_IPSEC_CBC_TDES_DECRYPT_MD5_APAD_FINAL	0x7A26	Perform the IPSec APAD finalization for decrypting in triple DES using CBC mode with MD5.
DPD_IPSEC_CBC_TDES_DECRYPT_MD5_FINAL	0x7A27	Perform the IPSec finalization for decrypting in triple DES using CBC mode with MD5.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA_INIT	0x7A28	Perform the IPSec initialization for decrypting in triple DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA_UPDATE	0x7A29	Perform the IPSec update for decrypting in triple DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA_APAD_FINAL	0x7A2A	Perform the IPSec APAD finalization for decrypting in triple DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA_FINAL	0x7A2B	Perform the IPSec finalization for decrypting in triple DES using CBC mode with SHA-1.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA256_ INIT	0x7A2C	Perform the IPSec initialization for decrypting in triple DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA256_ UPDATE	0x7A2D	Perform the IPSec update for decrypting in triple DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA256_ APAD_FINAL	0x7A2E	Perform the IPSec APAD finalization for decrypting in triple DES using CBC mode with SHA-256.
DPD_IPSEC_CBC_TDES_DECRYPT_SHA256_ FINAL	0x7A2F	Perform the IPSec finalization for decrypting in triple DES using CBC mode with SHA-256.

5.9.2 IPSEC_ECB_REQ

```
COMMON_REQ_PREAMBLE
unsigned long hashKeyBytes;
unsigned char* hashKeyData;
unsigned long cryptKeyBytes;
unsigned char* cryptKeyData;
unsigned long hashInDataBytes;
unsigned char* hashInData;
unsigned long inDataBytes;
unsigned char* inData;
unsigned long hashDataOutBytes;
unsigned char* hashDataOut;
unsigned char* cryptDataOut;
```

Dynamic and static channels are valid for this request. **NUM_IPSEC_ECB_DESC** defines the number of descriptors in the **DPD_IPSEC_ECB_GROUP** that use this request. **DPD_IPSEC_ECB_GROUP** (0x7100) defines the group for all descriptors in this request.

Table 42. IPSec_ECB_REQ Valid Descriptors (opId) for Dynamic Requests

Descriptors	Value	Function Description
`DPD_IPSEC_ECB_SDES_ENCRYPT_MD5_PAD`	0x7100	Perform IPSec encryption in single DES using ECB mode with MD5 padding.
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA_PAD`	0x7101	Perform IPSec encryption in single DES using ECB mode with SHA-1 padding.
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA256_PAD`	0x7102	Perform IPSec encryption in single DES using ECB mode with SHA-256 padding.
`DPD_IPSEC_ECB_SDES_DECRYPT_MD5_PAD`	0x7103	Perform IPSec process of decrypting in single DES using ECB mode with MD5 padding.
`DPD_IPSEC_ECB_SDES_DECRYPT_SHA_PAD`	0x7104	Perform IPSec decryption in single DES using ECB mode with SHA-1 padding.
`DPD_IPSEC_ECB_SDES_DECRYPT_SHA256_PAD`	0x7105	Perform IPSec decryption in single DES using ECB mode with SHA-256 padding.
`DPD_IPSEC_ECB_TDES_ENCRYPT_MD5_PAD`	0x7106	Perform IPSec encryption in triple DES using ECB mode with MD5 padding.
`DPD_IPSEC_ECB_TDES_ENCRYPT_SHA_PAD`	0x7107	Perform IPSec encryption in triple DES using ECB mode with SHA-1 padding.
`DPD_IPSEC_ECB_TDES_ENCRYPT_SHA256_PAD`	0x7108	Perform IPSec encryption in triple DES using ECB mode with SHA-256 padding.
`DPD_IPSEC_ECB_TDES_DECRYPT_MD5_PAD`	0x7109	Perform IPSec decryption in triple DES using ECB mode with MD5 padding.
`DPD_IPSEC_ECB_TDES_DECRYPT_SHA_PAD`	0x710A	Perform IPSec decryption in triple DES using ECB mode with SHA-1 padding.
`DPD_IPSEC_ECB_TDES_DECRYPT_SHA256_PAD`	0x710B	Perform IPSec decryption in triple DES using ECB mode with SHA-256 padding.
`DPD_IPSEC_ECB_SDES_ENCRYPT_MD5`	0x710C	Perform IPSec encryption in single DES using ECB mode with MD5.
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA`	0x710D	Perform IPSec encryption in single DES using ECB mode with SHA-1.
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA256`	0x710E	Perform IPSec encryption in single DES using ECB mode with SHA-256.
`DPD_IPSEC_ECB_SDES_DECRYPT_MD5`	0x710F	Perform IPSec decryption in single DES using ECB mode with MD5.
`DPD_IPSEC_ECB_SDES_DECRYPT_SHA`	0x7110	Perform IPSec decryption in single DES using ECB mode with SHA-1.
`DPD_IPSEC_ECB_SDES_DECRYPT_SHA256`	0x7111	Perform IPSec decryption in single DES using ECB mode with SHA-256.
`DPD_IPSEC_ECB_TDES_ENCRYPT_MD5`	0x7112	Perform IPSec encryption in triple DES using ECB mode with MD5.
`DPD_IPSEC_ECB_TDES_ENCRYPT_SHA`	0x7113	Perform IPSec encryption in triple DES using ECB mode with SHA-1.

Table 42. IPSec_ECB_REQ Valid Descriptors (opId) for Dynamic Requests (continued)

Descriptors	Value	Function Description
`DPD_IPSEC_ECB_TDES_ENCRYPT_SHA256`	0x7114	Perform IPSec encryption in triple DES using ECB mode with SHA-256.
`DPD_IPSEC_ECB_TDES_DECRYPT_MD5`	0x7115	Perform IPSec decryption in triple DES using ECB mode with MD5.
`DPD_IPSEC_ECB_TDES_DECRYPT_SHA`	0x7116	Perform IPSec decryption in triple DES using ECB mode with SHA-1.
`DPD_IPSEC_ECB_TDES_DECRYPT_SHA256`	0x7117	Perform IPSec decryption in triple DES using ECB mode with SHA-256.

`NUM_IPSEC_STATIC_ECB_DESC` defines the number of descriptors in the `DPD_IPSEC_STATIC_ECB_GROUP` that use this request. `DPD_IPSEC_STATIC_ECB_GROUP` (0x7B00) defines the group for all descriptors in this request.

Table 43. IPSec_ECB_REQ Valid Descriptors (opId) for Static Requests

Descriptors	Value	Function Description
`DPD_IPSEC_ECB_SDES_ENCRYPT_MD5_INIT`	0x7B00	Perform the IPSec initialization for encrypting in single DES using ECB mode with MD5.
`DPD_IPSEC_ECB_SDES_ENCRYPT_MD5_UPDATE`	0x7B01	Perform the IPSec update for encrypting in single DES using ECB mode with MD5.
`DPD_IPSEC_ECB_SDES_ENCRYPT_MD5_APAD_FINAL`	0x7B02	Perform the IPSec APAD finalization for encrypting in single DES using ECB mode with MD5
`DPD_IPSEC_ECB_SDES_ENCRYPT_MD5_FINAL`	0x7B03	Perform the IPSec finalization for encrypting in single DES using ECB mode with MD5.
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA_INIT`	0x7B04	Perform the IPSec initialization for encrypting in single DES using ECB mode with SHA-1
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA_UPDATE`	0x7B05	Perform the IPSec update for encrypting in single DES using ECB mode with SHA-1.
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA_APAD_FINAL`	0x7B06	Perform the IPSec APAD finalization for encrypting in single DES using ECB mode with SHA-1.
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA_FINAL`	0x7B07	Perform the IPSec finalization for encrypting in single DES using ECB mode with SHA-1.
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA256_INIT`	0x7B08	Perform the IPSec initialization for encrypting in single DES using ECB mode with SHA-256.
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA256_UPDATE`	0x7B09	Perform the IPSec update for encrypting in single DES using ECB mode with SHA-256.
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA256_APAD_FINAL`	0x7B0A	Perform the IPSec APAD finalization for encrypting in single DES using ECB mode with SHA-256.
`DPD_IPSEC_ECB_SDES_ENCRYPT_SHA256_FINAL`	0x7B0B	Perform the IPSec finalization for encrypting in single DES using ECB mode with SHA-256.
`DPD_IPSEC_ECB_SDES_DECRYPT_MD5_INIT`	0x7B0C	Perform the IPSec initialization for decrypting in single DES using ECB mode with MD5.

Table 43. IPSec_ECB_REQ Valid Descriptors (opId) for Static Requests (continued)

Descriptors	Value	Function Description
DPD_IPSEC_ECB_SDES_DECRYPT_MD5_UPDATE	0x7B0D	Perform the IPSec update for decrypting in single DES using ECB mode with MD5.
DPD_IPSEC_ECB_SDES_DECRYPT_MD5_APAD_FINAL	0x7B0E	Perform the IPSec APAD finalization for decrypting in single DES using ECB mode with MD5.
DPD_IPSEC_ECB_SDES_DECRYPT_MD5_FINAL	0x7B0F	Perform the IPSec finalization for decrypting in single DES using ECB mode with MD5.
DPD_IPSEC_ECB_SDES_DECRYPT_SHA_INIT	0x7B10	Perform the IPSec initialization for decrypting in single DES using ECB mode with SHA-1
DPD_IPSEC_ECB_SDES_DECRYPT_SHA_UPDATE	0x7B11	Perform the IPSec update for decrypting in single DES using ECB mode with SHA-1.
DPD_IPSEC_ECB_SDES_DECRYPT_SHA_APAD_FINAL	0x7B12	Perform the IPSec APAD finalization for decrypting in single DES using ECB mode with SHA-1.
DPD_IPSEC_ECB_SDES_DECRYPT_SHA_FINAL	0x7B13	Perform the IPSec finalization for decrypting in single DES using ECB mode with SHA-1.
DPD_IPSEC_ECB_SDES_DECRYPT_SHA256_INIT	0x7B14	Perform the IPSec initialization for decrypting in single DES using ECB mode with SHA-256.
DPD_IPSEC_ECB_SDES_DECRYPT_SHA256_UPDATE	0x7B15	Perform the IPSec update for decrypting in single DES using ECB mode with SHA-256.
DPD_IPSEC_ECB_SDES_DECRYPT_SHA256_APAD_FINAL	0x7B16	Perform the IPSec APAD finalization for decrypting in single DES using ECB mode with SHA-256.
DPD_IPSEC_ECB_SDES_DECRYPT_SHA256_FINAL	0x7B17	Perform the IPSec finalization for decrypting in single DES using ECB mode with SHA-256.
DPD_IPSEC_ECB_TDES_ENCRYPT_MD5_INIT	0x7B18	Perform the IPSec initialization for encrypting in triple DES using ECB mode with MD5.
DPD_IPSEC_ECB_TDES_ENCRYPT_MD5_UPDATE	0x7B19	Perform the IPSec update for encrypting in triple DES using ECB mode with MD5.
DPD_IPSEC_ECB_TDES_ENCRYPT_MD5_APAD_FINAL	0x7B1A	Perform the IPSec APAD finalization for encrypting in triple DES using ECB mode with MD5.
DPD_IPSEC_ECB_TDES_ENCRYPT_MD5_FINAL	0x7B1B	Perform the IPSec finalization for encrypting in triple DES using ECB mode with MD5.
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA_INIT	0x7B1C	Perform the IPSec initialization for encrypting in triple DES using ECB mode with SHA-1.
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA_UPDATE	0x7B1D	Perform the IPSec update for encrypting in triple DES using ECB mode with SHA-1.
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA_APAD_FINAL	0x7B1E	Perform the IPSec APAD finalization for encrypting in triple DES using ECB mode with SHA-1.
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA_FINAL	0x7B1F	Perform the IPSec finalization for encrypting in triple DES using ECB mode with SHA-1.
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA256_INIT	0x7B20	Perform the IPSec initialization for encrypting in triple DES using ECB mode with SHA-256.

Table 43. IPSec_ECB_REQ Valid Descriptors (opId) for Static Requests (continued)

Descriptors	Value	Function Description
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA256_UPDATE	0x7B21	Perform the IPSec update for encrypting in triple DES using ECB mode with SHA-256.
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA256_APAD_FINAL	0x7B22	Perform the IPSec APAD finalization for encrypting in triple DES using ECB mode with SHA-256.
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA256_FINAL	0x7B23	Perform the IPSec finalization for encrypting in triple DES using ECB mode with SHA-256.
DPD_IPSEC_ECB_TDES_DECRYPT_MD5_INIT	0x7B24	Perform the IPSec initialization for decrypting in triple DES using ECB mode with MD5.
DPD_IPSEC_ECB_TDES_DECRYPT_MD5_UPDATE	0x7B25	Perform the IPSec update for decrypting in triple DES using ECB mode with MD5.
DPD_IPSEC_ECB_TDES_DECRYPT_MD5_APAD_FINAL	0x7B26	Perform the IPSec APAD finalization for decrypting in triple DES using ECB mode with MD5.
DPD_IPSEC_ECB_TDES_DECRYPT_MD5_FINAL	0x7B27	Perform the IPSec finalization for decrypting in triple DES using ECB mode with MD5.
DPD_IPSEC_ECB_TDES_DECRYPT_SHA_INIT	0x7B28	Perform the IPSec initialization for decrypting in triple DES using ECB mode with SHA-1.
DPD_IPSEC_ECB_TDES_DECRYPT_SHA_UPDATE	0x7B29	Perform the IPSec update for decrypting in triple DES using ECB mode with SHA-1.
DPD_IPSEC_ECB_TDES_DECRYPT_SHA_APAD_FINAL	0x7B2A	Perform the IPSec APAD finalization for decrypting in triple DES using ECB mode with SHA-1.
DPD_IPSEC_ECB_TDES_DECRYPT_SHA_FINAL	0x7B2B	Perform the IPSec finalization for decrypting in triple DES using ECB mode with SHA-1.
DPD_IPSEC_ECB_TDES_DECRYPT_SHA256_INIT	0x7B2C	Perform the IPSec initialization for decrypting in triple DES using ECB mode with SHA-256
DPD_IPSEC_ECB_TDES_DECRYPT_SHA256_UPDATE	0x7B2D	Perform the IPSec update for decrypting in triple DES using ECB mode with SHA-256.
DPD_IPSEC_ECB_TDES_DECRYPT_SHA256_APAD_FINAL	0x7B2E	Perform the IPSec APAD finalization for decrypting in triple DES using ECB mode with SHA-256.
DPD_IPSEC_ECB_TDES_DECRYPT_SHA256_FINAL	0x7B2F	Perform the IPSec finalization for decrypting in triple DES using ECB mode with SHA-256.

5.9.3 IPSEC_AES_CBC_REQ

```
COMMON_REQ_PREAMBLE
unsigned long hashKeyBytes;
unsigned char* hashKeyData;
unsigned long cryptKeyBytes;
unsigned char* cryptKeyData;
unsigned long cryptCtxInBytes;
unsigned char* cryptCtxInData;
unsigned long hashInDataBytes;
unsigned char* hashInData;
unsigned long inDataBytes;
unsigned char* inData;
```

```
    unsigned char* cryptDataOut;
    unsigned long hashDataOutBytes;
    unsigned char* hashDataOut;
```

Dynamic channels are valid for this request. `NUM_IPSEC_AES_CBC_DESC` defines the number of descriptors within the `DPD_IPSEC_AES_CBC_GROUP` that use this request. `DPD_IPSEC_AES_CBC_GROUP` (0x8000) defines the group for all descriptors within this request.

Table 44. IPSec_AES_CBC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_IPSEC_AES_CBC_ENCRYPT_MD5_APAD`	0x8000	Perform IPSec encryption in AES using CBC mode with MD5 auto padding.
`DPD_IPSEC_AES_CBC_ENCRYPT_SHA_APAD`	0x8001	Perform IPSec encryption in AES using CBC mode with SHA-1 auto padding.
`DPD_IPSEC_AES_CBC_ENCRYPT_SHA256_APAD`	0x8002	Perform IPSec encryption in AES using CBC mode with SHA-256 auto padding.
`DPD_IPSEC_AES_CBC_ENCRYPT_MD5`	0x8003	Perform IPSec encryption in AES using CBC mode with MD5
`DPD_IPSEC_AES_CBC_ENCRYPT_SHA`	0x8004	Perform IPSec encryption in AES using CBC mode with SHA-1.
`DPD_IPSEC_AES_CBC_ENCRYPT_SHA256`	0x8005	Perform IPSec encryption in AES using CBC mode with SHA-256.
`DPD_IPSEC_AES_CBC_DECRYPT_MD5_APAD`	0x8006	Perform IPSec decryption in AES using CBC mode with MD5 auto padding.
`DPD_IPSEC_AES_CBC_DECRYPT_SHA_APAD`	0x8007	Perform IPSec decryption in AES using CBC mode with SHA-1 auto padding.
`DPD_IPSEC_AES_CBC_DECRYPT_SHA256_APAD`	0x8008	Perform IPSec decryption in AES using CBC mode with SHA-256 auto padding.
`DPD_IPSEC_AES_CBC_DECRYPT_MD5`	0x8009	Perform IPSec decryption in AES using CBC mode with MD5
`DPD_IPSEC_AES_CBC_DECRYPT_SHA`	0x800A	Perform IPSec decryption in AES using CBC mode with SHA-1.
`DPD_IPSEC_AES_CBC_DECRYPT_SHA256`	0x800B	Perform IPSec decryption in AES using CBC mode with SHA-256.
`DPD_IPSEC_AES_CBC_DECRYPT_MD5_APAD_RESTK`	0x800C	Perform IPSec decryption in AES using CBC mode with MD5 auto padding and restacking.
`DPD_IPSEC_AES_CBC_DECRYPT_SHA_APAD_RESTK`	0x800D	Perform IPSec decryption in AES using CBC mode with SHA-1 auto padding and restacking.
`DPD_IPSEC_AES_CBC_DECRYPT_SHA256_APAD_RESTK`	0x800E	Perform IPSec decryption in AES using CBC mode with SHA-256 auto padding and restacking.
`DPD_IPSEC_AES_CBC_DECRYPT_MD5_RESTK`	0x800F	Perform IPSec decryption in AES using CBC mode with MD5 and restacking.

Table 44. IPSec_AES_CBC_REQ Valid Descriptors (opId) (continued)

Descriptors	Value	Function Description
DPD_IPSEC_AES_CBC_DECRYPT_SHA_RESTK	0x8010	Perform IPSec decryption in AES using CBC mode with SHA-1 and restacking.
DPD_IPSEC_AES_CBC_DECRYPT_SHA256_ RESTK	0x8011	Perform IPSec decryption in AES using CBC mode with SHA-256 and restacking.

5.9.4 IPSEC_AES_ECB_REQ

```
COMMON_REQ_PREAMBLE
unsigned long hashKeyBytes;
unsigned char* hashKeyData;
unsigned long cryptKeyBytes;
unsigned char* cryptKeyData;
unsigned long hashInDataBytes;
unsigned char* hashInData;
unsigned long inDataBytes;
unsigned char* inData;
unsigned char* cryptDataOut;
unsigned long hashDataOutBytes;
unsigned char* hashDataOut;
```

Dynamic channels are valid for this request. **NUM_IPSEC_AES_ECB_DESC** defines the number of descriptors in the **DPD_IPSEC_AES_ECB_GROUP** that use this request. **DPD_IPSEC_AES_ECB_GROUP** (0x8100) defines the group for all descriptors in this request.

Table 45. IPSec_AES_ECB_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_IPSEC_AES_ECB_ENCRYPT_MD5_APAD	0x8100	Perform IPSec encryption in AES using ECB mode with MD5 auto padding.
DPD_IPSEC_AES_ECB_ENCRYPT_SHA_APAD	0x8101	Perform IPSec encryption in AES using ECB mode with SHA-1 auto padding.
DPD_IPSEC_AES_ECB_ENCRYPT_SHA256_APAD	0x8102	Perform IPSec encryption in AES using ECB mode with SHA-256 auto padding.
DPD_IPSEC_AES_ECB_ENCRYPT_MD5	0x8103	Perform IPSec encryption in AES using ECB mode with MD5
DPD_IPSEC_AES_ECB_ENCRYPT_SHA	0x8104	Perform IPSec encryption in AES using ECB mode with SHA-1.
DPD_IPSEC_AES_ECB_ENCRYPT_SHA256	0x8105	Perform IPSec encryption in AES using ECB mode with SHA-256.
DPD_IPSEC_AES_ECB_DECRYPT_MD5_APAD	0x8106	Perform IPSec decryption in AES using ECB mode with MD5 auto padding.
DPD_IPSEC_AES_ECB_DECRYPT_SHA_APAD	0x8107	Perform IPSec decryption in AES using ECB mode with SHA-1 auto padding.
DPD_IPSEC_AES_ECB_DECRYPT_SHA256_APAD	0x8108	Perform IPSec decryption in AES using ECB mode with SHA-256 auto padding.

Table 45. IPSec_AES_ECB_REQ Valid Descriptors (opId) (continued)

Descriptors	Value	Function Description
`DPD_IPSEC_AES_ECB_DECRYPT_MD5`	0x8109	Perform IPSec decryption in AES using ECB mode with MD5.
`DPD_IPSEC_AES_ECB_DECRYPT_SHA`	0x810A	Perform IPSec decryption in AES using ECB mode with SHA-1.
`DPD_IPSEC_AES_ECB_DECRYPT_SHA256`	0x810B	Perform IPSec decryption in AES using ECB mode with SHA-256.
`DPD_IPSEC_AES_ECB_DECRYPT_MD5_APAD_RESTK`	0x810C	Perform IPSec decryption in AES using ECB mode with MD5 auto padding and restacking.
`DPD_IPSEC_AES_ECB_DECRYPT_SHA_APAD_RESTK`	0x810D	Perform IPSec decryption in AES using ECB mode with SHA-1 auto padding and restacking.
`DPD_IPSEC_AES_ECB_DECRYPT_SHA256_APAD_RESTK`	0x810E	Perform IPSec decryption in AES using ECB mode with SHA-256 auto padding and restacking.
`DPD_IPSEC_AES_ECB_DECRYPT_MD5_RESTK`	0x810F	Perform IPSec decryption in AES using ECB mode with MD5 and restacking.
`DPD_IPSEC_AES_ECB_DECRYPT_SHA_RESTK`	0x8110	Perform IPSec decryption in AES using ECB mode with SHA-1 and restacking.
`DPD_IPSEC_AES_ECB_DECRYPT_SHA256_RESTK`	0x8111	Perform IPSec decryption in AES using ECB mode with SHA-256 and restacking.

5.10 802.11 Protocol Requests

5.10.1 CCMP_REQ

```
COMMON_REQ_PREAMBLE
unsigned long  keyBytes;
unsigned char *keyData;
unsigned long  ctxBytes;
unsigned char *context;
unsigned long  FrameDataBytes;
unsigned char *FrameData;
unsigned long  cryptDataBytes;
unsigned char *cryptDataOut;
unsigned long  MICBytes;
unsigned char *MICData;
```

`NUM_CCMP_DESC` defines the number of descriptors within the **`DPD_CCMP_GROUP`** that use this request. **`DPD_CCMP_GROUP`** (0x6500) defines the group for all descriptors in this request.

Table 46. CCMP_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
`DPD_802_11_CCMP_OUTBOUND`	`0x6500`	Process an outbound CCMP packet
`DPD_802_11_CCMP_INBOUND`	`0x8101`	Process an inbound CCMP packet

`CCMP_REQ` on SEC1 differs slightly from **`CCMP_REQ`** for SEC2 in that the latter can support a separate packet header and thus has an extra member to point to that header. Note this critical difference if you are porting code between the two platforms.

6 Sample Code

The following sections provide sample codes for DES and IPSec.

6.1 DES Sample

```
/* define the User Structure */
DES_LOADCTX_CRYPT_REQ desencReq;
.
.
.
/* fill the User Request structure with appropriate pointers */
desencReq.opId                    = DPD_TDES_CBC_ENCRYPT_SA_LDCTX_CRYPT ;
desencReq.channel                 = 0;     /* dynamic channel */
desencReq.notify                  = (void*) notifyDes; /* callback function */
desencReq.notify_on_error         = (void*) notifyDes; /* callback in case of
                                                        errors only */
desencReq.status                  = 0;
desencReq.ivBytes                 = 8;         /* input iv length */
desencReq.ivData                  = iv_in;     /* pointer to input iv */
desencReq.keyBytes                = 24;        /* key length */
desencReq.keyData                 = DesKey;    /* pointer to key */
desencReq.inBytes                 = packet_length;         /* data length */
desencReq.inData                        = DesData;         /* pointer to data */
desencReq.outData                 = desEncResult;          /* pointer to results */
desencReq.nextReq                 = 0;               /* no descriptor chained */

/* call the driver */
status = Ioctl(device, IOCTL_PROC_REQ, &desencReq);

/* First Level Error Checking */
if (status != 0) {
      .
      .
  }
.
.
.


void notifyDes (void)
{
/* Second Level Error Checking */
if (desencReq.status != 0) {
      .
      .
  }
.
.
)
```

6.2 IPSec Sample

```
/* define User Requests structures */
IPSEC_CBC_REQ     ipsecReq;
.
.
.
.


/* Ipsec dynamic descriptor triple DES with SHA-1 authentication */
ipsecReq.opId          = DPD_IPSEC_CBC_TDES_ENCRYPT_SHA_PAD;
ipsecReq.channel             = 0;
ipsecReq.notify              = (void *) notifyFunc;
ipsecReq.notify_on_error     = (void *) notifyFunc;
ipsecReq.status              = 0;
ipsecReq.hashKeyBytes        = 16; /* key length for HMAC SHA-1 */
ipsecReq.hashKeyData         = authKey; /* pointer to HMAC Key */
ipsecReq.cryptCtxInBytes     = 8;    /* length of input iv */
ipsecReq.cryptCtxInData      = in_iv; /* pointer to input iv */
ipsecReq.cryptKeyBytes       = 24; /* DES key length */
ipsecReq.cryptKeyData        = EncKey; /* pointer to DES key */
ipsecReq.hashInDataBytes     = 8; /* length of data to be hashed only */
ipsecReq.hashInData          = PlainText; /* pointer to data to be
                                             hashed only */
ipsecReq.inDataBytes         = packet_length-8;  /* length of data to be
                                                   hashed and encrypted */
ipsecReq.inData              = &PlainText[8];     /* pointer to data to be
                                                   hashed and encrypted */
ipsecReq.cryptDataOut        = Result; /* pointer to encrypted results */
ipsecReq.hashDataOutBytes    = 20;       /* length of output digest */
ipsecReq.hashDataOut         = digest;   /* pointer to output digest */
ipsecReq.nextReq             = 0;        /* no chained requests */

/* call the driver */
status = Ioctl(device, IOCTL_PROC_REQ, &ipsecReq);

/* First Level Error Checking */
if (status != 0) {
      .
      .
      .
  }
.
.
.


void notifyFunc (void)
{
/* Second Level Error Checking */
if (ipsecReq.status != 0) {
      .
      .
      .
  }
.
.
)
```

7 Linux Environment

This section describes the driver adaptation to and interaction with the Linux operating system as applied to PowerPC processors.

7.1 Installation

The SEC1 driver installs into Linux as a loadable module. To build the driver as a module, it must be installed into the kernel source tree for inclusion in the kernel build process. The makefile with the distribution assumes this inclusion. As delivered, this directory is defined as `[kernelroot]/drivers/sec1`

After the driver source is installed and the kernel source (and modules) are built, module dependency lists updated, and the built objects are installed in the target file system, the driver, (named `sec1drv.o`) is ready for loading. Kernel processes can directly call the driver functionality. In contrast, user processes must use the kernel I/O interface to issue driver requests. User processes open the device as a file with the `open()` system call to get a file descriptor, and then they make requests through `ioctl()`. Thus, the system needs a device file created to assign a name to the device.

The driver functions as a `char` device in the target system. As shipped, the driver assumes that the device major number is assigned dynamically and that the minor number is always zero because only one instance of the driver is supported. The device naming inode can be created manually in a development setting, or it can be driven by a script that runs after the driver module loads and before a user attempts to open a path to the driver. If the module is loaded with a dynamically assigned major number of 254 (look for `sec1` in `/proc/devices`), the shell command to accomplish this normally appears as follows:

```
$ mknod c 254 0 /dev/sec1
```

With this task accomplished, user tasks can make requests to the driver under the device name `/dev/sec1`.

8 Driver Operation in Kernel Mode

Operation of the SEC1 device in kernel mode is relatively straightforward. After the driver module is loaded, which initializes the device, direct calls to the `ioctl()` entry (named `SEC1_ioctl` in the driver) can be made. The first two arguments can effectively be ignored. In kernel mode, request completion can be handled through the standard use of notification callbacks in the request. The example suite available with the driver shows how this is accomplished with a mutex that the callback releases to allow the request to complete. However, the caller can use any other type of event mechanism as preferred. Logical to physical memory space translation is handled internally to the driver.

9 Driver Operation in User Mode

Operation of the SEC1 device in user mode is slightly more complex than in kernel mode. The transition from user to kernel memory space creates two complications for user mode operation:

1. User memory buffers cannot be passed directly to the driver; instead, the user must allocate and place data into a kernel memory buffer through `SEC1_MALLOC`, `SEC1_FREE`, `SEC1_COPYFROM`, and `SEC1_COPYTO` requests (see Section 4.1, “I/O Control Codes”).

Use *extreme* caution in transferring memory this way; the caller can easily corrupt kernel memory space, causing instability in the target system.

2. Standard notification callbacks cannot work because the routines to perform the callback are in user memory space and cannot safely execute from kernel mode. In their place, standard POSIX signals can be used to indicate I/O completion by placing the process ID of the user task in the notification members of the request and flagging `NOTIFY_IS_PID` in the `notifyFlags` member. The driver uses `SIGUSR1` to indicate normal request completions and `SIGUSR2` to indicate error completions.

The example suite available with the driver illustrates the contrast between the two different application environments. Within the `testAll.c` file is a set of functions that shows the difference between the two operations. Building the example testing application with `__KERNEL__` on (building a kernel mode test) shows the installation and usage of standard completion callbacks and a mutex for interlock. Conversely, building the example testing application with `USERMODE` turned on shows the installation of signal handlers and their proper setup. In `USERMODE`, this example also shows one way to handle the user-to-kernel memory transition through the use of three functions for transferring user buffers to and from kernel memory.

10 VxWorks Environment

This section describes the installation of the SEC1 security engine software drivers, BSP integration, and distribution archives.

10.1 Installing the Software Drivers

To install the software drivers, extract the archive containing the driver source files into a suitable installation directory. If the driver and tests are to be part of a standard VxWorks source tree, place them was follows:

Driver:	`$(WIND_BASE)/target/src/drv/crypto`
Tests:	`$(WIND_BASE)/target/src/drv/crypto/test`

After the modules are installed, the driver image can be built.

10.2 Building the Interface Modules

In the installation instructions that follow, the variables listed here are used:

VxWorks Interface Module Variables

Variable	Definition
CpuFamily	Specifies the target CPU family, such as PPC603
ToolChain	Specifies the tools, such as `gnu`
SecurityProcessor	Specifies the target security engine, which should be `SEC1` for this driver

If you are building code for the MPC855 processor, the name `DUET` must be defined and passed to the compiler with a –D option. The following steps are used to build drivers and/or the driver test and exercise code:

1. Go to the command prompt or shell.
2. Execute `torVars` to set up the Tornado command line build environment, as follows:

```
Run make in the driver or test installation directory by use of the following command:
make CPU=cpuFamily TOOL=toolChain SP=securityProcessor
(example: make CPU=PPC603 TOOL=gnu SP=SEC1)
```

10.3 Integrating the BSP

After the modules are built, they should be linked directly with the user board support package (BSP), to become integral part of the board image. In VxWorks, the `sysLib.c` file contains the initialization functions, the memory/address space functions, and the bus interrupt functions. Call the function `SEC1DriverInit` directly from `sysLib.c`. The security processor is then initialized at board startup with all the other devices present on the board.

11 Porting the Driver

This section describes how to port the driver to other operating systems or environments. This driver has been ported to both VxWorks and the Linux operating systems. Most internal functionality is independent of the constructs of a specific operating system, but there are interface boundaries that must be addressed. Only the following files in the driver source distribution contain dependencies on operating system components:

- `Sec1Driver.h`
- `sec1_init.c`
- `sec1_io.c`

11.1 Header Files

The `Sec1Driver.h` header file is meant to be local (private) to the driver itself, and it includes all needed operating system header files and casts a series of macros for specific system calls. Of particular interest, this header casts local equivalent macros for:

`malloc`	Allocate a block of system memory with the operating system’s heap allocation mechanism.
`free`	Return a block of memory to the system heap
`semGive`	Release a mutex semaphore
`semTake`	Capture and hold a mutex semaphore
`__vpa`	Translate a logical address to a physical address for hardware DMA (if both are equivalent, does nothing).

11.2 C Source Files

- `sec1_init.c`. Performs the basic initialization of the device and the driver, finds the base address of the hardware, and saves it in `IOBaseAddress` for later reference.

  For Linux, this file also contains references to register/unregister the driver as a kernel module and to manage its usage/link count.
- `sec1_io.c`. Contains functions to establish:
  - Channel interlock semaphores (`IOInitSemaphores`)
  - ISR message queue (`IOInitQs`)
  - Driver service function registration with the operating system (`IORegisterDriver`)
  - ISR connection/disconnection (`IOConnectInterrupt`)

11.3 Interrupt Service Routine (ISR)

The ISR queues processing completion result messages onto the `IsrMsgQId` queue. `ProcessingComplete()` pends on this message queue. When a message is received, the completion task executes the appropriate callback routine based on the result of the processing. When the end-user application prepares the request for execution, callback functions can be defined for nominal processing as well as error case processing. If the callback function is set to `NULL` when the request is prepared, no callback function executes. These routines execute as part of the device driver, so any constraints on the device driver are also placed on the callback routines.

11.4 Conditional Compilation

See the makefile for specifics on the default build of the driver.

11.5 Debug Messaging

The driver includes a `DBG` define that allows for debug message output to the developer console. If defined in the driver build, debug messages are sent from various components in the driver to the console. Messages come from various sections of the driver, and a bitmask is kept in a driver global variable so that the developer can turn message sources on or off as required. This variable is named `SEC1DebugLevel`, and it contains an ORed combination of any of the following bits:

`DBGTXT_SETRQ`	Messages from request setup operations (new requests inbound from the application).
`DBGTXT_SVCRQ`	Messages from servicing device responses (ISR/deferred service routine handlers) outbound to the application.
`DBGTXT_INITDEV`	Messages from the device/driver initialization process.
`DBGTXT_DPDSHOW`	Shows the content of a constructed DPD before it is handed to the security core.
`DBGTXT_INFO`	Shows a short banner at device initialization describing the driver and hardware version.

In normal driver operation (not in a development setting), the DBG definition should be left undefined for best performance.

11.6 Distribution Archive

For this release, the distribution archive consists of the source files listed in Table 47. You may wish to reorganize header file locations to make them consistent with the file location conventions appropriate for their system configuration.

Table 47. Distribution Archive Source Files

Header	Description
`Sec1.h`	Primary public header file for all users of the driver
`Sec1Driver.h`	Driver/Hardware interfaces, private to the driver itself
`Sec1_Descriptors.h`	DPD type definitions
`Sec1Notify.h`	Structures for ISR/main thread communication
`sec1_dpd_Table.h`	DPD construction constants
`sec1_cha.c`	CHA mapping and management
`sec1_dpd.c`	DPD construction functionality
`sec1_init.c`	Device/driver initialization code
`sec1_io.c`	Basic register I/O primitives
`sec1_ioctl.c`	Operating system interfaces
`sec1_request.c`	Request/response management
`sec1isr.c`	Interrupt service routine

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
1-800-521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: SEC1SWUG
Rev. 0
5/2006