MPC18xSWUG
Rev. 0, 9/2003

MPC184/MPC185 Security Co-Processor Software User's Guide

This document assumes that the user has a basic familiarity with the MPC18x architecture and theory of operation. A review of the MPC184 or MPC185 user's manual is suggested prior to beginning application development.

This document describes the software drivers provided to access the MPC184 and MPC185 security co-processors, and how to use them. It covers the following topics:

PRELIMINARY—SUBJECT TO CHANGE WITHOUT NOTICE

Table 1 contains acronyms and abbreviations that are used in this user's manual.

Table 1. Acronyms and Abbreviations

Term	Meaning
AESA	AES accelerator—This term is synonymous with AESU in the *MPC18x User's Manual* and other documentation.
AFHA	ARC-4 hardware accelerator—This term is synonymous with AFEU in the *MPC18x User's Manual* and other documentation.
APAD	Autopad—The MDHA will automatically pad incomplete message blocks out to 512 bits when APAD is enabled.
ARC-4	Encryption algorithm compatible with the RC-4 algorithm developed by RSA, Inc.
Auth	Authentication
CBC	Cipher block chaining—An encryption mode commonly used with block ciphers.
CHA	Crypto hardware accelerator—This term is synonymous with 'execution unit' in the *MPC18x User's Manual* and other documentation.
CTX	Context
DESA	DES accelerator—This term is synonymous with DEU in the *MPC18x User's Manual* and other documentation.
DPD	Data packet descriptor
ECB	Electronic code book—An encryption mode less commonly used with block ciphers.
EU	Execution unit
HMAC	Hashed message authentication code
IDGS	Initialize digest
IPSec	Internet protocol security
ISR	Interrupt service routine
KEA	Kasumi encryption acceleration
MD	Message digest
MDHA	Message digest hardware accelerator—This term is synonymous with MDEU in the *MPC18x User's Manual* and other documentation.
OS	Operating system
PK	Public key
PKHA	Public key hardware accelerator—This term is synonymous with PKEU in the *MPC18x User's Manual* and other documentation.
RDK	Restore decrypt key—An AESA option to re-use an existing expanded AES decryption key.
RNGA	Random number generator accelerator
SDES	Single DES
TEA	Transfer error acknowledge
TDES	Triple DES
VxWorks	Operating systems provided by VxWorks Company.

1 Overview

The MCP18x device driver controls the communications to the MCP184 and MPC185 chips from a variety of processors. The MPC184 can operate in PCI or 8xx bus mode (PowerQUICC I). The MPC185 operates in 60x bus mode. Both chips are memory mapped and are accessed by applications via the drivers’ system or device memory.

The device driver is written in ANSI C. An attempt has been made to write a device driver that is independent from the operating system agnostic, whenever possible. The operating system dependencies are well identified and Section 6, “Porting,” addresses them.

The device driver has been tested in VxWorks 5.4 and Linux (*Hard Hat Linux 2.0 Journeyman Edition by Monte Vista—Kernel Version 2.4.2*).

Throughout this document, VxWorks terminology is used for the discussion of information. Linux specific differences are addressed in Section 6, “Porting.”

The driver’s interface is implemented through the **`ioctl`** function call. The functions or requests made through this interface can be broken down into specific components, which include miscellaneous requests and process requests. The miscellaneous requests are all other requests not related to process requests. These include get_status, malloc, and free. Malloc and free are available for Linux only.

Process requests make up the majority of the requests and all are executed using the same **`ioctl`** function. In Section 4.6, “Process Request Structures,” the structures used to define these requests are described.

Throughout the document, the acronyms CHA (crypto hardware accelerator) and EU (execution unit) are used interchangeably. Both acronyms indicate the device’s functional block that performs the crypto functions required. For further details on the device see the Hardware Reference Manual.

2 Device Driver Components

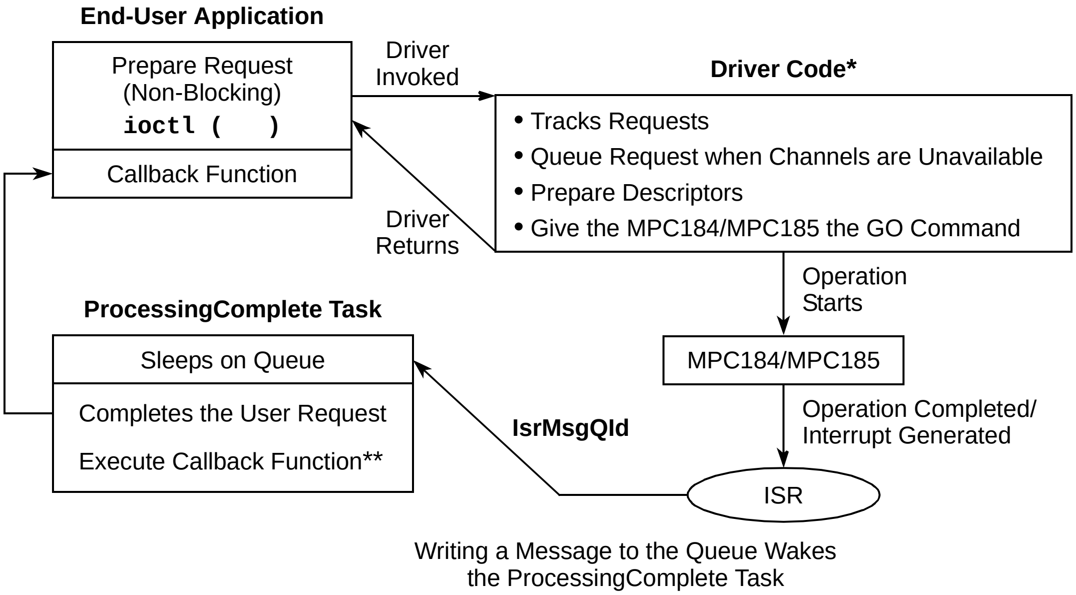
This section is provided to help users understand the internal structure of the device driver.

2.1 Device Driver Structure

Internally, the driver is structured in four basic parts:

- Initialization
- I/O request
- Interrupt service routine
- Processing complete

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



* Driver runs under the context of the end-user task.
** If no callback function is defined, no callback takes place.

Figure 1. Internal Driver Architecture

2.2 Driver Initialization Routine

The driver initialization routine includes both OS-specific (VxWorks and Linux) and hardware-specific (MPC185 60x, MPC184 PCI, and MPC184 8xx) initialization.

The steps taken by the driver initialization routine are as follows:

- Find the security co-processor and save the device memory map starting address in **`IOBaseAddress`** (based on the actual device)
- Initialize the security co-processor registers
  - Controller registers
  - Channel registers
  - EU registers

- Initialize the driver internal variables
- Initialize the ChannelAssignments table
  - The device driver will maintain this structure with state information for each channel and user request. A Mutex semaphore protects this structure so multiple tasks are prevented from interfering with each other.
- Initialize the internal queue
  - A queue that holds requests to be dispatched when channels are available. The queue will hold up to 16 requests. The driver will reject requests when the queue is full.
- ProcessingComplete is spawned and pends on the IsrMsgQId which serves as the interface between the interrupt service routine and this deferred task.

2.3 Request Dispatch Routine

The request dispatch routine provides the **`ioctl`** interface to the device driver. It uses the I/O control code to identify what function is to be called and dispatches the appropriate routine to process the request.

The **`ioctl`** function runs under the context of the end-user application. The **`ioctl`** function invokes the driver code. The driver performs a number of tasks that include tracking requests, queuing requests when the requested channel is unavailable, preparing descriptors, and writing the descriptor’s address to the appropriate channel; in effect giving the security co-processor the GO command to begin processing the request. The **`ioctl`** function returns to the end-user application without waiting for the co-processor to complete.

2.4 Process Request Routine

The process request routine translates the request into a sequence of one or more data packet descriptors (DPD) and starts the operation. Dynamic requests will be queued if no channels are available. Static requests will fail and return an error if the requested channel is not available or busy.

2.5 Interrupt Service Routine

When processing is completed in the security co-processor 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 task.

2.6 ProcessingComplete Task

The ProcessingComplete task completes the request outside of the interrupt service routine as a separate task at a lower priority. This task depends on the IsrMsgQId queue and processes messages put into the queue by the interrupt service routine. This task will determine which request is complete and notify the corresponding calling task. It will then check the process request queue and schedule any queued requests.

3 User Interface

3.1 End-User Application

The end-user application populates the request structure with the appropriate information. These structures are described in Section 4, “Global Definitions,” and include the operation ID, channel, callback routines (success and error), and data. Once the request is prepared, the end-user application calls the **ioctl** function and passes the prepared request. The **ioctl** function is a standard function call used by many drivers and present in different operating systems. It follows the standard format:

```
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 MPC18x user request structure (see Section 4.6, “Process Request Structures”), that contains the information needed by the driver to perform the function requested.

The following is a list of guidelines to be followed by the end-user application when preparing a request structure:

- The first member of every request structure is an operation ID. The operation ID is used 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 set to zero to indicate dynamic mode or to a valid (non-zero) channel number to indicate static mode. For process requests that only work in static mode, the channel should be set to a valid (non-zero) channel number.
- All process request structures have a status member. This value is filled in by the device driver when the interrupt for the operation occurs and it reflects the type of interrupt. The valid values for this status member are DONE (normal status) or ERROR (error status).
- All process request structures have two notify members, done and error. These notify members are used by the device driver to notify the application when its request has been completed.
- All process request structures have a next request member. This allows the application to chain multiple process requests together.
- It is the application’s choice to use the callback function or to poll the status member.

3.2 Static vs. 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 benefit for using static mode is faster internal EU execution. The drawbacks of using static mode are: the channel and EU cannot be used by anyone else and the overhead of assign/release of channel/EU.

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

The code used 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);
.
.
```

3.3 Error Handling

Due to the asynchronous nature of the device/driver, there are two main sources of errors:

- Syntax or logic. These are returned in the **`status`** member of the ‘user request’ argument and as a return code from **`ioctl`** function. Errors of this type are detected by drivers, 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.

Consequently the end-user application needs two levels of error checking, the first one after the return from the **`ioctl`** function, and the second one after the completion of the request. The second level is possible only if the request was done with at least the **`notify_on_error`** member of the user request structure. If the notification/callback function has not been requested, this level of error will be lost.

A code example of the two levels of errors are as follows:

```
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

4.1 I/O Control Codes

The I/O control code is the second argument in the **ioctl** function.

Internally, these values (as shown in Table 2), 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 **MPC18x_IOCTL_INDEX** and has a value of 0x0800.

Table 2. Second and Third Arguments in the ioctl Function

I/O Control Code (Second Argument in ioctl Function)	Value	Third Argument in ioctl Function
IOCTL_PROC_REQ	0x0801	Request Structure *
IOCTL_GET_STATUS	0x0802	STATUS_REQ *
IOCTL_RESERVE_CHANNEL_STATIC	0x0804	unsigned long *channel
IOCTL_RESERVE_CHANNEL_MANUAL [1]	0x0805	MPC18x_RESERVE_MANUAL *
IOCTL_ASSIGN_CHA	0x0806	unsigned long cha
IOCTL_RELEASE_CHA	0x0807	unsigned long cha
IOCTL_RELEASE_CHANNEL	0x0808	unsigned long *channel
The following are used for LINUX only		
IOCTL_MALLOC	0x080D	MALLOC_REQ *
IOCTL_FREE	0x080E	MALLOC_REQ *
IOCTL_COPYFROM	0x080F	MALLOC_REQ *
IOCTL_COPYTO	0x0810	MALLOC_REQ *

[1] This control code is used exclusively in debug/slave mode. The drivers make it available but do not use it.

Note: The * in the third column of the above table indicates a pointer.

4.2 Channel Definitions

The **NUM_CHANNELS** define is used to specify the number of channels in the MCP18x (see Table 3). If not specified it will be set to a value of 4 as a default.

Table 3. Channel Defines

Define	Description	Value For	
		MCP185	MCP184 (pci or 8xx)
NUM_AFHAS	Number of ARC4 CHAs	1	1
NUM_DESAS	Number of DES CHAs	2	1
NUM_MDHAS	Number of MD CHAs	2	1
NUM_RNGAS	Number of RNG CHAs	1	1
NUM_PKHAS	Number of PK CHAs	2	1
NUM_AESAS	Number of AESA CHAs	2	1
NUM_KEA	Number of KEA CHAs	1	0

The **NUM_CHAS** define contains the total number of crypto hardware accelerators (CHAs) in the MPC18x and is simply defines as the sum of the individual channels in Table 3.

The device that is used is defined as the macro **VxWorksDrvName** regardless of whether or not VxWorks is being used. For the MCP185 this is set to '/dev/mpc185' and for the MCP184 this is set to '/dev/mpc184.'

4.3 Request Operation Id (opId) Masks

Operation Ids can be broken down into two parts, the group or type of request and the request index or descriptor within a group or type (see Table 4). This is provided to help understand the structuring of the opIds. It is not specifically needed within a user application.

Table 4. 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 Code Definitions

Table 5 provides a complete list of the error status results that may be returned to the callback routines.

Table 5. Callback Error Status Return Code

Define	Description	Value
`MPC18x_SUCCESS`	Successful completion of call	0
`MPC18x_MEMORY_ALLOCATION`	Error due to memory allocation	0xE004FFFF
`MPC18x_INVALID_CHANNEL`	Error due to invalid channel	0xE004FFFE
`MPC18x_INVALID_CHA_TYPE`	Error due to invalid CHA type	0xE004FFFD
`MPC18x_INVALID_OPERATION_ID`	Error due to invalid OpId	0xE004FFFC
`MPC18x_CHANNEL_NOT_AVAILABLE`	Error due to channel not being available	0xE004FFFB
`MPC18x_CHA_NOT_AVAILABLE`	Error due to CHA not being available	0xE004FFFA
`MPC18x_INVALID_LENGTH`	Error due to invalid request length	0xE004FFF9
`MPC18x_OUTPUT_BUFFER_ALIGNMENT`	Error due to output buffer alignment	0xE004FFF8
`MPC18x_ADDRESS_PROBLEM`	Error due to address not being allocated	0xE004FFF6
`MPC18x_INSUFFICIENT_REQS`	Error due to insufficient `req_list` entries	0xE004FFF5
`MPC18x_STATIC_CHANNEL_BUSY`	Error due to channel already handling static request	0xE004FFF3
`MPC18x_CHA_ERROR`	Error due to problem with CHA	0xE004FFF2
`MPC18x_NULL_REQUEST`	Error due to NULL request	0xE004FFF1
`MPC18x_REQUEST_TIMED_OUT`	Error due to request timing out	0xE004FFF0
`MPC18x_MALLOC_FAILED`	Error due to memory mgmt malloc failing	0xE004FFEF
`MPC18x_FREE_FAILED`	Error due to memory mgmt free failing	0xE004FFEE
`MPC18x_PARITY_SYSTEM_ERROR`	Parity Errordetected on the bus	0xE004FFED
`MPC18x_INCOMPLETE_POINTER`	Error due to partial pointer	0xE004FFEC
`MPC18x_TEA_ERROR`	A transfer error has occurred	0xE004FFEB
`MPC18x_UNKNOWN_ERROR`	Any other unrecognized error	0xE004FFEA
`MPC18x_IO_CARD_NOT_FOUND`	Error due to MPC18x card not being found	–1000
`MPC18x_IO_MEMORY_ALLOCATE_ERROR`	Error due to insufficient resources	–1001
`MPC18x_IO_IO_ERROR`	Error due to I/O configuration	–1002
`MPC18x_IO_VXWORKS_DRIVER_TABLE_ADD_ERROR`	Error due to VxWorks not being able to add driver to table	–1003
`MPC18x_IO_INTERRUPT_ALLOCATE_ERROR`	Error due to interrupt allocation error	–1004

Table 5. Callback Error Status Return Code (continued)

Define	Description	Value
MPC18x_CANNOT_SETUP_BAR0_ERROR	Error due to VxWorks not being able to setup the base address in pciGetBAR0()	–1008
MPC18x_VXWORKS_CANNOT_CREATE_QUEUE	Error due to VxWorks not being able to create the ISR queue in IOInitQs()	–1009
MPC18x_CANCELLED_REQUEST	Error due to canceled request	–1010
MPC18x_INVALID_ADDRESS	Error due to a NULL request	–1011

4.5 Miscellaneous Request Structures

4.5.1 MALLOC_REQ (Linux Only) Structure

This structure is used in Linux for allocation requests to the driver. In Linux the user memory space is different than the kernel memory space. Due to this limitation, in order to send/receive requests to the kernel, the user must request memory space from the kernel (malloc) and then perform a copy of the contents in user memory to/from kernel memory. As with all mallocs, the user must free this memory. The MALLOC_REQ structure is used to facilitate this process.

```
unsigned long     sz;
void          *ptr;
char          *to;
char          *from;
```

sz—contains the number of bytes to allocate. Zero means to use the default. A value of zero can be used to avoid fragmentation.

ptr—pointer to the address that is to be returned by a call to malloc or a pointer to an address that is to be freed when by a call to free.

to—pointer to the address that is the destination of the data during a copy memory request.

from—pointer to the address that is the source of the data during a copy memory request.

This request will be used by user applications running in ‘user space’ only.

User applications running in kernel space do not need to allocate memory using this function.

Following is an example of how to use this function:

```
DES_LOADCTX_CRYPT_REQ desencReq;
MALLOC_REQ mem;
      .
      .
      /* copy iv data to kernel memory */
      mem.sz = 8;
      status = Ioctl(device, IOCTL_MALLOC, &mem);
      desencReq.inIvData = mem.ptr;
      mem.from = iv_in;
      mem.to = mem.ptr;
      status = Ioctl(device, IOCTL_COPYFROM, &mem);
      /* copy key data to kernel memory */
      mem.sz = 8;
      status = Ioctl(device, IOCTL_MALLOC, &mem);
      desencReq.keyData = mem.ptr;
      mem.from = desKey;
      mem.to = mem.ptr;
      status = Ioctl(device, IOCTL_COPYFROM, &mem);
      /* copy data to kernel memory */
      mem.sz = len;
      status = Ioctl(device, IOCTL_MALLOC, &mem);
      desencReq.inData = mem.ptr;
      mem.from = desData;
      mem.to = mem.ptr;
      mem.sz = len;
      status = Ioctl(device, IOCTL_COPYFROM, &mem);

      /* malloc kernel output memory */
      mem.sz = len;
      status = Ioctl(device, IOCTL_MALLOC, &mem);
      desDecResult = mem.ptr;
      status = Ioctl(device, IOCTL_MALLOC, &mem);
      desEncResult = mem.ptr;
      mem.sz = 8;
      status = Ioctl(device, IOCTL_MALLOC, &mem);
      desCtxOut = mem.ptr;
.
.
      if (status = Ioctl(device, IOCTL_PROC_REQ, &desencReq))
.
.
```

4.5.2 STATUS_REQ Structure

Structure used to indicate the state of the MPC18x as well as the driver. Returned as a pointer by GetStatus( ) and imbedded in all requests. Each element is a copy of the contents of the same register in the MCP18x driver. This structure is also known as MPC18x_STATUS through a typedef.

```
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 FreeAfhas;
unsigned long FreeDesas;
unsigned long FreeMdhas;
unsigned long FreePkhas;
unsigned long FreeAesas;
unsigned long FreeKeas;
unsigned long BlockSize;
```

4.5.3 MPC18x_NOTIFY_ON_ERROR_CTX Structure

Structure returned to the notify_on_error callback routine that was setup in the initial process request. This structure contains the original request structure as well as an error and driver status.

```
unsigned long errorcode;    // Error that the request generated
void          *request;     // Pointer to original request
STATUS_REQ    driverstatus;// Detailed information as to 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—detailed information as to the state of the hardware and the driver at the time of an error

4.6 Process Request Structures

All process request structures contain the same header information, which begins with the GENERIC_REQ structure, though no such structure is explicitly defined.

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
```

opId—operation Id which identifies what type of request this is.

channel—identifies the channel to be used for the request. A value of zero indicates that the request is dynamic. If a request is specifically for static channels, zero is not valid.

notify—pointer to the notify routine that will be called when the request has completed successfully.

pNotifyCtx—pointer to context area to be passed back through the notify routine.

notify_on_error—pointer to the notify on error routine that will be called when the request has completed unsuccessfully.

ctxNotifyOnErr—context area that is filled in by the driver when there is an error.

status—will contain the returned status of request.

nextReq—pointer to next request which allows for multiple request to be linked together and sent via a single ioclt function call.

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

4.6.1 Random Number Generation Request Structures

The following section provides structure definitions for the random number generation process request.

4.6.1.1 RNG_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_RNG_GROUP** that use this request.

DPD_RNG_GROUP (0x1000) defines the group for all descriptors within this request.

Table 6. RNG_REQ Valid Descriptor (opId)

Descriptor	Value	Function Description
DPD_RNG_GETRN	0x1000	Get random number

4.6.2 DES Process Request Structures

The following sections provide structure definitions for DES process requests.

4.6.2.1 DES_LOADCTX_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_DES_SA_LDCTX_GROUP** that use this request.

DPD_DES_SA_LDCTX_GROUP (0x2000) defines the group for all descriptors within this request.

Table 7. 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 single DES using CBC mode
DPD_SDES_CBC_DECRYPT_SA_LDCTX	0x2001	Load context from a static channel to decrypt in single DES using CBC mode
DPD_SDES_ECB_ENCRYPT_SA_LDCTX	0x2002	Load context from a static channel to encrypt in single DES using ECB mode
DPD_SDES_ECB_DECRYPT_SA_LDCTX	0x2003	load context from a static channel to decrypt in single DES using ECB mode
DPD_TDES_CBC_ENCRYPT_SA_LDCTX	0x2004	Load context from a static channel to encrypt in 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

4.6.2.2 DES_LOADCTX_CRYPT_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_DES_SA_LDCTX_CRYPT_GROUP** that use this request.

DPD_DES_SA_LDCTX_CRYPT_GROUP (0x2100) defines the group for all descriptors within this request.

Table 8. 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
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

4.6.2.3 DES_CRYPT_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_DES_SA_CRYPT_GROUP** that use this request.

DPD_DES_SA_CRYPT_GROUP (0x2200) defines the group for all descriptors within this request.

Table 9. 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 single DES using CBC mode
DPD_SDES_CBC_DECRYPT_SA_CRYPT	0x2201	Decrypt data in a static channel in single DES using CBC mode
DPD_SDES_ECB_ENCRYPT_SA_CRYPT	0x2202	Encrypt data in a static channel in single DES using ECB mode
DPD_SDES_ECB_DECRYPT_SA_CRYPT	0x2203	Decrypt data in a static channel in single DES using ECB mode
DPD_TDES_CBC_ENCRYPT_SA_CRYPT	0x2204	Encrypt data in a static channel in triple DES using CBC mode
DPD_TDES_CBC_DECRYPT_SA_CRYPT	0x2205	Decrypt data in a static channel in triple DES using CBC mode
DPD_TDES_ECB_ENCRYPT_SA_CRYPT	0x2206	Encrypt data in a static channel in triple DES using ECB mode
DPD_TDES_ECB_DECRYPT_SA_CRYPT	0x2207	Decrypt data in a static channel in triple DES using ECB mode

4.6.2.4 DES_CRYPT_GETCTX_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within 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 10. 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 that was encrypted in single DES using CBC mode
DPD_SDES_CBC_DECRYPT_SA_CRYPT_ULCTX	0x2301	Get context from a static channel that was decrypted in single DES using CBC mode
DPD_SDES_ECB_ENCRYPT_SA_CRYPT_ULCTX	0x2302	Get context from a static channel that was encrypted in single DES using ECB mode
DPD_SDES_ECB_DECRYPT_SA_CRYPT_ULCTX	0x2303	Get context from a static channel that was decrypted in single DES using ECB mode
DPD_TDES_CBC_ENCRYPT_SA_CRYPT_ULCTX	0x2304	Get context from a static channel that was encrypted in triple DES using CBC mode
DPD_TDES_CBC_DECRYPT_SA_CRYPT_ULCTX	0x2305	Get context from a static channel that was decrypted in triple DES using CBC mode
DPD_TDES_ECB_ENCRYPT_SA_CRYPT_ULCTX	0x2306	Get context from a static channel that was encrypted in triple DES using ECB mode
DPD_TDES_ECB_DECRYPT_SA_CRYPT_ULCTX	0x2307	Get context from a static channel that was decrypted in triple DES using ECB mode

4.6.2.5 DES_GETCTX_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_DES_SA_ULCTX_GROUP** that use this request.

DPD_DES_SA_ULCTX_GROUP (0x2400) defines the group for all descriptors within this request.

Table 11. 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

4.6.2.6 DES_LOADCTX_CRYPT_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 12. 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_DECRYPT_SA_LDCTX	0x2501	Load encrypted context from a dynamic channel to decrypt in single DES using CBC mode
DPD_TDES_CBC_CTX_ENCRYPT	0x2502	Load encrypted context from a dynamic channel to decrypt in single DES using CBC mode
DPD_TDES_CBC_CTX_DECRYPT	0x2503	Load encrypted context from a dynamic channel to encrypt in single DES using CBC mode

4.6.2.7 DES_CRYPT_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 13. 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

4.6.3 ARC4 Process Request Structures

The following sections provide structure definitions for ARC4 process requests.

4.6.3.1 ARC4_NEWCTX_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 14. ARC4_NEWCTX_STATIC_REQ Valid Descriptor (opId)

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

4.6.3.2 ARC4_LOADCTX_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 15. 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

4.6.3.3 ARC4_CRYPT_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 16. ARC4_CRYPTO_STATIC_REQ Valid Descriptor (opId)

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

4.6.3.4 ARC4_CRYPT_GETCTX_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 17. 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

4.6.3.5 ARC4_LOADCTX_CRYPT_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 18. 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 then get the resulting context

4.6.3.6 ARC4_LOADKEY_CRYPT_UNLOADCTX_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 19. 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 then get the resulting context

4.6.4 Hash Request Structures

The following sections provide structure definitions for hash requests.

4.6.4.1 HASH_LOADCTX_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 20. HASH_LOADCTX_STATIC_REQ Valid Descriptors (opId)

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

4.6.4.2 HASH_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within this request.

Table 21. 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 22. 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

4.6.4.3 HASH_GETCTX_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 23. HASH_GETCTX_STATIC_REQ Valid Descriptors (opId)

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

4.6.4.4 HASH_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 24. HASH_REQ Valid Descriptors (0x4400) (opId)

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

NUM_MDHA_PAD_DESC defines the number of descriptors within 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 within this request.

Table 25. HASH_REQ Valid Descriptors (0x4500) (opId)

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

4.6.5 HMAC Request Structures

The following sections provide structure definitions for HMAC requests.

4.6.5.1 HMAC_PAD_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_HMAC_SA_PAD_GROUP** that use this request.

DPD_HMAC_SA_PAD_GROUP (0x4600) defines the group for all descriptors within this request.

Table 26. 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 using an SHA-256 hash algorithm with padding
DPD_HMAC_SA_MD5_PAD	0x4601	Perform a HMAC operation on a static channel to using an MD5 hash algorithm with padding
DPD_HMAC_SA_SHA_PAD	0x4602	Perform a HMAC operation on a static channel to using an SHA-1 hash algorithm with padding
DPD_HMAC_SA_SHA256_PAD_IDGS	0x4603	Perform a HMAC operation on a static channel to using an SHA-256 hash algorithm with IDGS padding
DPD_HMAC_SA_MD5_PAD_IDGS	0x4604	Perform a HMAC operation on a static channel to using an MD5 hash algorithm with IDGS padding
DPD_HMAC_SA_SHA_PAD_IDGS	0x4605	Perform a HMAC operation on a static channel to using an SHA-1 hash algorithm with IDGS padding

4.6.5.2 HMAC_PAD_HASH_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_HMAC_SA_PAD_HASH_GROUP** that use this request.

DPD_HMAC_SA_PAD_HASH_GROUP (0x4700) defines the group for all descriptors within this request.

Table 27. HMAC_PAD_HASH_STATIC_REQ Valid Descriptors (opId)

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

4.6.5.3 HMAC_PAD_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_HASH_LDCTX_HMAC_ULCTX_GROUP** that use this request.

DPD_HASH_LDCTX_HMAC_ULCTX_GROUP (0x4A00) defines the group for all descriptors within this request.

Table 28. HMAC_PAD_REQ Valid Descriptors (opId)

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

4.6.6 AES Request Structures

The following section provides structure definitions for AES requests.

4.6.6.1 AESA_CRYPT_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_AESA_CRYPT_GROUP** that use this request.

DPD_AESA_CRYPT_GROUP (0x6000) defines the group for all descriptors within this request.

Table 29. AESA_CRYPT_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_AESA_CBC_ENCRYPT_CRYPT	0x6000	Perform encryption in AESA using CBC mode
DPD_AESA_CBC_DECRYPT_CRYPT	0x6001	Perform decryption in AESA using CBC mode
DPD_AESA_CBC_DECRYPT_CRYPT_RDK	0x6002	Perform decryption in AESA using CBC mode with RDK
DPD_AESA_ECB_ENCRYPT_CRYPT	0x6003	Perform encryption in AESA using ECB mode
DPD_AESA_ECB_DECRYPT_CRYPT	0x6004	Perform decryption in AESA using ECB mode
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

4.6.7 Kasumi Request Structures

The following section provides structure definitions for Kasumi requests.

4.6.7.1 KEA_CRYPT_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 */
```

Dynamic channels are valid for this request. A channel value of zero is valid.

NUM_KEA_CRYPT_DESC defines the number of descriptors within the **DPD_KEA_CRYPT_GROUP** that use this request.

DPD_KEA_CRYPT_GROUP (0xA000) defines the group for all descriptors within this request.

Table 30. KEA_CRYPT_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_KEA_f8_UPLINK_CRYPT	0xA000	Perform an f8 key exchange algorithm uplink
DPD_KEA_f8_DOWNLINK_CRYPT	0xA001	Perform an f8 key exchange algorithm downlink
DPD_KEA_f9_UPLINK_CRYPT	0xA002	Perform an f9 key exchange algorithm uplink
DPD_KEA_f9_DOWNLINK_CRYPT	0xA003	Perform an f9 key exchange algorithm downlink

4.6.8 Integer Public Key Request Structures

The following sections provide structure definitions for integer public key requests.

4.6.8.1 MOD_EXP_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_MM_SA_EXP_GROUP** that use this request.

DPD_MM_SA_EXP_GROUP (0x5000) defines the group for all descriptors within this request.

Table 31. 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

4.6.8.2 MOD_EXP_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_MM_LDCTX_EXP_ULCTX_GROUP** that use this request.

DPD_MM_LDCTX_EXP_ULCTX_GROUP (0x5100) defines the group for all descriptors within this request.

Table 32. MOD_EXP_REQ Valid Descriptor (opId)

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

4.6.8.3 MOD_R2MODN_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_MM_LDCTX_R2MODN_ULCTX_GROUP** that use this request.

DPD_MM_LDCTX_R2MODN_ULCTX_GROUP (0x5200) defines the group for all descriptors within this request.

Table 33. MOD_R2MODN_REQ Valid Descriptor (opId)

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

4.6.8.4 MOD_RRMODP_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_MM_LDCTX_RRMODP_ULCTX_GROUP** that use this request.

DPD_MM_LDCTX_RRMODP_ULCTX_GROUP (0x5300) defines the group for all descriptors within this request.

Table 34. 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 a RRMODP operation

4.6.8.5 MOD_2OP_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_MM_LDCTX_2OP_ULCTX_GROUP** that use this request.

DPD_MM_LDCTX_2OP_ULCTX_GROUP (0x5400) defines the group for all descriptors within this request.

Table 35. MOD_2OP_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_MM_LDCTX_MUL1_ULCTX	0x5400	Load context in a dynamic channel and return the resulting context from a MUL1 operation
DPD_MM_LDCTX_MUL2_ULCTX	0x5401	Load context in a dynamic channel and return the resulting context from a MUL2 operation
DPD_MM_LDCTX_ADD_ULCTX	0x5402	Load context in a dynamic channel and return the resulting context from a ADD operation
DPD_MM_LDCTX_SUB_ULCTX	0x5403	Load context in a dynamic channel and return the resulting context from a SUB operation
DPD_POLY_LDCTX_A0_B0_MUL1_ULCTX	0x5404	Load context in 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 in 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 in 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 in 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 in 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 in 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 in a dynamic channel and return the resulting context from an A2-to-B0 MUL1 operation

Table 35. MOD_2OP_REQ Valid Descriptors (opId) (continued)

Descriptors	Value	Function Description
DPD_POLY_LDCTX_A2_B0_MUL2_ULCTX	0x540B	Load context in 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 in 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 in 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 in 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 in 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 in 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 in 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 in 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 in a dynamic channel and return the resulting context from an A1-to-B1 MUL1 operation
DPD_POLY_LDCTX_A1_B1_MUL2_ULCTX	0x5414	Load context in 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 in 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 in 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 in 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 in 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 in 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 in 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 in 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 in 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 in 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 in a dynamic channel and return the resulting context from an A0-to-B2ADD operation
DPD_POLY_LDCTX_A1_B2_MUL1_ULCTX	0x541F	Load context in a dynamic channel and return the resulting context from an A1-to-B2 MUL1 operation

Table 35. MOD_2OP_REQ Valid Descriptors (opId) (continued)

Descriptors	Value	Function Description
`DPD_POLY_LDCTX_A1_B2_MUL2_ULCTX`	0x5420	Load context in 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 in 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 in 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 in 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 in 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 in 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 in 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 in a dynamic channel and return the resulting context from an A3-to-B2 ADD operation
`DPD_POLY_LDCTX_A0_B3_MUL1_ULCTX`	0x5428	Load context in 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 in 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 in 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 in 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 in 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 in 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 in 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 in 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 in 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 in 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 in 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 in a dynamic channel and return the resulting context from an A3-to-B3 ADD operation

4.6.8.6 MOD_CLR_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_MM_SA_CLR_GROUP** that use this request.

DPD_MM_SA_CLR_GROUP (0x5A00) defines the group for all descriptors within this request.

Table 36. MOD_CLR_STATIC_REQ Valid Descriptor (opId)

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

4.6.9 ECC Public Key Request Structures

The following sections provide structure definitions for ECC public key requests.

4.6.9.1 ECC_LOADPOINTK_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_EC_SA_LOADCTX_GROUP** that use this request.

DPD_EC_SA_LOADCTX_GROUP (0x5500) defines the group for all descriptors within this request.

Table 37. 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

4.6.9.2 ECC_LOADPARAM_PMULT_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_EC_SA_kP_GROUP** that use this request.

DPD_EC_SA_kP_GROUP (0x5600) defines the group for all descriptors within this request.

Table 38. 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
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

4.6.9.3 ECC_GETRESULT_STATIC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_EC_SA_ULCTX_GROUP** that use this request.

DPD_EC_SA_ULCTX_GROUP (0x5700) defines the group for all descriptors within this request.

Table 39. 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

4.6.9.4 ECC_POINT_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_EC_LDCTX_kP_ULCTX_GROUP** that use this request.

DPD_EC_LDCTX_kP_ULCTX_GROUP (0x5800) defines the group for all descriptors within this request.

Table 40. ECC_POINT_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_EC_FP_AFF_LDCTX_kP_ULCTX	0x5800	Load context in a dynamic channel to using an electronic codebook for the FP AFF then get the resulting P multiplier context
DPD_EC_FP_PROJ_LDCTX_kP_ULCTX	0x5801	Load context in a dynamic channel to using an electronic codebook for the FP project then get the resulting P multiplier context
DPD_EC_F2M_AFF_LDCTX_kP_ULCT	0x5802	Load context in a dynamic channel to using an electronic codebook for the F2M AFF then get the resulting P multiplier context
DPD_EC_F2M_PROJ_LDCTX_kP_ULCTX	0x5803	Load context in a dynamic channel to using an electronic codebook for the F2M project then get the resulting P multiplier context
DPD_EC_FP_LDCTX_ADD_ULCT	0x5804	Load context in a dynamic channel to using an electronic codebook for the FP then get the resulting context from an add operation
DPD_EC_FP_LDCTX_DOUBLE_ULCTX	0x5805	Load context in a dynamic channel to using an electronic codebook for the FP then get the resulting context from a double operation

Table 40. ECC_POINT_REQ Valid Descriptors (opId) (continued)

Descriptors	Value	Function Description
`DPD_EC_F2M_LDCTX_ADD_ULCTX`	0x5806	Load context in a dynamic channel to using an electronic codebook for the F2M then get the resulting context from an add operation
`DPD_EC_F2M_LDCTX_DOUBLE_ULCTX`	0x5807	Load context in a dynamic channel to using an electronic codebook for the F2M then get the resulting context from a double operation

4.6.9.5 ECC_2OP_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 41. ECC_2OP_REQ Valid Descriptor (opId)

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

4.6.10 IPSec Request Structures

The following sections provide structure definitions for IPSec requests.

4.6.10.1 IPSEC_CBC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_IPSEC_CBC_GROUP** that use this request.

DPD_IPSEC_CBC_GROUP (0x7000) defines the group for all descriptors within this request.

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

Descriptors	Value	Function Description
DPD_IPSEC_CBC_SDES_ENCRYPT_MD5_PAD	0x7000	Perform the IPSec process of encrypting in single DES using CBC mode with MD5 padding
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA_PAD	0x7001	Perform the IPSec process of encrypting in single DES using CBC mode with SHA-1 padding
DPD_IPSEC_CBC_SDES_ENCRYPT_SHA256_PAD	0x7002	Perform the IPSec process of encrypting in single DES using CBC mode with SHA-256 padding
DPD_IPSEC_CBC_SDES_DECRYPT_MD5_PAD	0x7003	Perform the IPSec process of decrypting in single DES using CBC mode with MD5 padding
DPD_IPSEC_CBC_SDES_DECRYPT_SHA_PAD	0x7004	Perform the IPSec process of decrypting in single DES using CBC mode with SHA-1 padding
DPD_IPSEC_CBC_SDES_DECRYPT_SHA256_PAD	0x7005	Perform the IPSec process of decrypting in single DES using CBC mode with SHA-256 padding
DPD_IPSEC_CBC_TDES_ENCRYPT_MD5_PAD	0x7006	Perform the IPSec process of encrypting in triple DES using CBC mode with MD5 padding

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

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

NUM_IPSEC_STATIC_CBC_DESC defines the number of descriptors within the **DPD_IPSEC_STATIC_CBC_GROUP** that use this request.

DPD_IPSEC_STATIC_CBC_GROUP (0x7A00) defines the group for all descriptors within this request.

Table 43. 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
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

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

Descriptors	Value	Function Description
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
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

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

Descriptors	Value	Function Description
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

4.6.10.2 IPSEC_ECB_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_IPSEC_ECB_GROUP** that use this request.

DPD_IPSEC_ECB_GROUP (0x7100) defines the group for all descriptors within this request.

Table 44. IPSec_ECB_REQ Valid Descriptors (opId) for Dynamic Requests

Descriptors	Value	Function Description
DPD_IPSEC_ECB_SDES_ENCRYPT_MD5_PAD	0x7100	Perform the IPSec process of encrypting in single DES using ECB mode with MD5 padding
DPD_IPSEC_ECB_SDES_ENCRYPT_SHA_PAD	0x7101	Perform the IPSec process of encrypting in single DES using ECB mode with SHA-1 padding
DPD_IPSEC_ECB_SDES_ENCRYPT_SHA256_PAD	0x7102	Perform the IPSec process of encrypting in single DES using ECB mode with SHA-256 padding
DPD_IPSEC_ECB_SDES_DECRYPT_MD5_PAD	0x7103	Perform the IPSec process of decrypting in single DES using ECB mode with MD5 padding
DPD_IPSEC_ECB_SDES_DECRYPT_SHA_PAD	0x7104	Perform the IPSec process of decrypting in single DES using ECB mode with SHA-1 padding
DPD_IPSEC_ECB_SDES_DECRYPT_SHA256_PAD	0x7105	Perform the IPSec process of decrypting in single DES using ECB mode with SHA-256 padding
DPD_IPSEC_ECB_TDES_ENCRYPT_MD5_PAD	0x7106	Perform the IPSec process of encrypting in triple DES using ECB mode with MD5 padding
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA_PAD	0x7107	Perform the IPSec process of encrypting in triple DES using ECB mode with SHA-1 padding
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA256_PAD	0x7108	Perform the IPSec process of encrypting in triple DES using ECB mode with SHA-256 padding
DPD_IPSEC_ECB_TDES_DECRYPT_MD5_PAD	0x7109	Perform the IPSec process of decrypting in triple DES using ECB mode with MD5 padding

Table 44. IPSec_ECB_REQ Valid Descriptors (opId) for Dynamic Requests (continued)

Descriptors	Value	Function Description
DPD_IPSEC_ECB_TDES_DECRYPT_SHA_PAD	0x710A	Perform the IPSec process of decrypting in triple DES using ECB mode with SHA-1 padding
DPD_IPSEC_ECB_TDES_DECRYPT_SHA256_PAD	0x710B	Perform the IPSec process of decrypting in triple DES using ECB mode with SHA-256 padding
DPD_IPSEC_ECB_SDES_ENCRYPT_MD5	0x710C	Perform the IPSec process of encrypting in single DES using ECB mode with MD5
DPD_IPSEC_ECB_SDES_ENCRYPT_SHA	0x710D	Perform the IPSec process of encrypting in single DES using ECB mode with SHA-1
DPD_IPSEC_ECB_SDES_ENCRYPT_SHA256	0x710E	Perform the IPSec process of encrypting in single DES using ECB mode with SHA-256
DPD_IPSEC_ECB_SDES_DECRYPT_MD5	0x710F	Perform the IPSec process of decrypting in single DES using ECB mode with MD5
DPD_IPSEC_ECB_SDES_DECRYPT_SHA	0x7110	Perform the IPSec process of decrypting in single DES using ECB mode with SHA-1
DPD_IPSEC_ECB_SDES_DECRYPT_SHA256	0x7111	Perform the IPSec process of decrypting in single DES using ECB mode with SHA-256
DPD_IPSEC_ECB_TDES_ENCRYPT_MD5	0x7112	Perform the IPSec process of encrypting in triple DES using ECB mode with MD5
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA	0x7113	Perform the IPSec process of encrypting in triple DES using ECB mode with SHA-1
DPD_IPSEC_ECB_TDES_ENCRYPT_SHA256	0x7114	Perform the IPSec process of encrypting in triple DES using ECB mode with SHA-256
DPD_IPSEC_ECB_TDES_DECRYPT_MD5	0x7115	Perform the IPSec process of decrypting in triple DES using ECB mode with MD5
DPD_IPSEC_ECB_TDES_DECRYPT_SHA	0x7116	Perform the IPSec process of decrypting in triple DES using ECB mode with SHA-1
DPD_IPSEC_ECB_TDES_DECRYPT_SHA256	0x7117	Perform the IPSec process of decrypting in triple DES using ECB mode with SHA-256

NUM_IPSEC_STATIC_ECB_DESC defines the number of descriptors within the **DPD_IPSEC_STATIC_ECB_GROUP** that use this request.

DPD_IPSEC_STATIC_ECB_GROUP (0x7B00) defines the group for all descriptors within this request.

Table 45. 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
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

Table 45. IPSec_ECB_REQ Valid Descriptors (opId) for Static Requests (continued)

Descriptors	Value	Function Description
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
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

Table 45. IPSec_ECB_REQ Valid Descriptors (opId) for Static Requests (continued)

Descriptors	Value	Function Description
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

4.6.10.3 IPSEC_AES_CBC_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 46. IPSec_AES_CBC_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_IPSEC_AES_CBC_ENCRYPT_MD5_APAD	0x8000	Perform the IPSec process of encrypting in AES using CBC mode with MD5 auto padding
DPD_IPSEC_AES_CBC_ENCRYPT_SHA_APAD	0x8001	Perform the IPSec process of encrypting in AES using CBC mode with SHA-1 auto padding
DPD_IPSEC_AES_CBC_ENCRYPT_SHA256_APAD	0x8002	Perform the IPSec process of encrypting in AES using CBC mode with SHA-256 auto padding
DPD_IPSEC_AES_CBC_ENCRYPT_MD5	0x8003	Perform the IPSec process of encrypting in AES using CBC mode with MD5
DPD_IPSEC_AES_CBC_ENCRYPT_SHA	0x8004	Perform the IPSec process of encrypting in AES using CBC mode with SHA-1
DPD_IPSEC_AES_CBC_ENCRYPT_SHA256	0x8005	Perform the IPSec process of encrypting in AES using CBC mode with SHA-256
DPD_IPSEC_AES_CBC_DECRYPT_MD5_APAD	0x8006	Perform the IPSec process of decrypting in AES using CBC mode with MD5 auto padding
DPD_IPSEC_AES_CBC_DECRYPT_SHA_APAD	0x8007	Perform the IPSec process of decrypting in AES using CBC mode with SHA-1 auto padding
DPD_IPSEC_AES_CBC_DECRYPT_SHA256_APAD	0x8008	Perform the IPSec process of decrypting in AES using CBC mode with SHA-256 auto padding

Table 46. IPSec_AES_CBC_REQ Valid Descriptors (opId) (continued)

Descriptors	Value	Function Description
DPD_IPSEC_AES_CBC_DECRYPT_MD5	0x8009	Perform the IPSec process of decrypting in AES using CBC mode with MD5
DPD_IPSEC_AES_CBC_DECRYPT_SHA	0x800A	Perform the IPSec process of decrypting in AES using CBC mode with SHA-1
DPD_IPSEC_AES_CBC_DECRYPT_SHA256	0x800B	Perform the IPSec process of decrypting in AES using CBC mode with SHA-256
DPD_IPSEC_AES_CBC_DECRYPT_MD5_APAD_RESTK	0x800C	Perform the IPSec process of decrypting in AES using CBC mode with MD5 auto padding and restacking
DPD_IPSEC_AES_CBC_DECRYPT_SHA_APAD_RESTK	0x800D	Perform the IPSec process of decrypting in AES using CBC mode with SHA-1 auto padding and restacking
DPD_IPSEC_AES_CBC_DECRYPT_SHA256_APAD_RESTK	0x800E	Perform the IPSec process of decrypting in AES using CBC mode with SHA-256 auto padding and restacking
DPD_IPSEC_AES_CBC_DECRYPT_MD5_RESTK	0x800F	Perform the IPSec process of decrypting in AES using CBC mode with MD5 and restacking
DPD_IPSEC_AES_CBC_DECRYPT_SHA_RESTK	0x8010	Perform the IPSec process of decrypting in AES using CBC mode with SHA-1 and restacking
DPD_IPSEC_AES_CBC_DECRYPT_SHA256_RESTK	0x8011	Perform the IPSec process of decrypting in AES using CBC mode with SHA-256 and restacking

4.6.10.4 IPSEC_AES_ECB_REQ

```
unsigned long opId;
unsigned long channel;
PMPC18x_NOTIFY_ROUTINE notify;
PMPC18x_NOTIFY_CTX pNotifyCtx;
PMPC18x_NOTIFY_ON_ERROR_ROUTINE notify_on_error;
MPC18x_NOTIFY_ON_ERROR_CTX ctxNotifyOnErr;
int status;
void* nextReq;
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 within the **DPD_IPSEC_AES_ECB_GROUP** that use this request.

DPD_IPSEC_AES_ECB_GROUP (0x8100) defines the group for all descriptors within this request.

Table 47. IPSec_AES_ECB_REQ Valid Descriptors (opId)

Descriptors	Value	Function Description
DPD_IPSEC_AES_ECB_ENCRYPT_MD5_APAD	0x8100	Perform the IPSec process of encrypting in AES using ECB mode with MD5 auto padding
DPD_IPSEC_AES_ECB_ENCRYPT_SHA_APAD	0x8101	Perform the IPSec process of encrypting in AES using ECB mode with SHA-1 auto padding
DPD_IPSEC_AES_ECB_ENCRYPT_SHA256_APAD	0x8102	Perform the IPSec process of encrypting in AES using ECB mode with SHA-256 auto padding
DPD_IPSEC_AES_ECB_ENCRYPT_MD5	0x8103	Perform the IPSec process of encrypting in AES using ECB mode with MD5
DPD_IPSEC_AES_ECB_ENCRYPT_SHA	0x8104	Perform the IPSec process of encrypting in AES using ECB mode with SHA-1
DPD_IPSEC_AES_ECB_ENCRYPT_SHA256	0x8105	Perform the IPSec process of encrypting in AES using ECB mode with SHA-256
DPD_IPSEC_AES_ECB_DECRYPT_MD5_APAD	0x8106	Perform the IPSec process of decrypting in AES using ECB mode with MD5 auto padding
DPD_IPSEC_AES_ECB_DECRYPT_SHA_APAD	0x8107	Perform the IPSec process of decrypting in AES using ECB mode with SHA-1 auto padding
DPD_IPSEC_AES_ECB_DECRYPT_SHA256_APAD	0x8108	Perform the IPSec process of decrypting in AES using ECB mode with SHA-256 auto padding
DPD_IPSEC_AES_ECB_DECRYPT_MD5	0x8109	Perform the IPSec process of decrypting in AES using ECB mode with MD5

Table 47. IPSec_AES_ECB_REQ Valid Descriptors (opId) (continued)

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

5 Sample Code

The following sections provide sample codes for DES and IPSec.

5.1 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) {
      .
      .
  }
.
.
)
```

5.2 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) {
      .
      .
      .
  }
.
.
)
```

6 Porting

The following sections describe the main operating system specific concept of semaphores and the interrupt service routine (ISR), and the required files for the code, configuration, and external variables.

6.1 Include Files

The interface module code uses the file **vxWorks.h**. This includes standard VxWorks type includes. Using this code in another system will require redefining those types for that system. For example, the type UINT32 should be defined as a 32-bit unsigned integer on your target system. In addition, for the semaphore mechanism the file **semLib.h** is included, to provide definitions for the semaphore type (SEM_ID) and the semaphore function calls. This will need to be replaced by the appropriate files depending on the communication mechanism used for the port.

For Linux, **vxWorks.h** has been replaced by **RTLinux.h**. Several drivers also use the ANSI standard string functions specified in **string.h.**

The first type of semaphore is the Mutex, which is used to use protect driver structures during critical updates. There are three calls involved with this semaphore:

- VxWorks
  - semMCreate—This call creates the semaphore for later use.
  - semTake—This call takes the semaphore, preventing another task from using the resource.
  - semGive—This call gives the semaphore, allowing another process to use the resource.
- Linux kernel
  - init_MUTEX—This call creates the semaphore for later use.
  - up—This call takes the semaphore, preventing another task from using the resource.
  - down_timeout—This call gives the semaphore, allowing another process to use the resource.
- Linux user
  - sem_init—This call creates the semaphore for later use.
  - sem_post—This call takes the semaphore, preventing another task from using the resource.
  - sem_wait—This call gives the semaphore, allowing another process to use the resource.

The semaphores used in the code are as follows:

- SEM_ID ChannelAssignSemId;
- SEM_ID QueueSemId;

The second types of semaphores are binary semaphores that are used by the test programs to wait for the ISR. In other multitasking operating systems, these calls should be replaced with the appropriate semaphore calls for the specific operating system.

There are three calls involved with these semaphores:

- VxWorks
  - semBCreate—This call creates the semaphore for later use.
  - semTake—This call is used in mainline code to wait on the semaphore. It usually has a timeout associated with it.
  - semGive—This call gives the semaphore allowing the mainline task to continue.

- Linux kernel
  - init_MUTEX_LOCKED—This call creates the semaphore for later use.
  - up—This call takes the semaphore, preventing another task from using the resource.
  - down_timeout—This call gives the semaphore, allowing another process to use the resource.
- Linux user
  - sem_init—This call creates the semaphore for later use.
  - sem_post—This call takes the semaphore, preventing another task from using the resource.
  - sem_wait—This call gives the semaphore, allowing another process to use the resource.

6.2 Interrupt Service Routine

As shown in Figure 1, the ISR will queue processing result messages onto the IsrMsgQId queue. The ProcessingComplete task pends on this message queue. When a message is received this task will execute the appropriate callback routine based on the result of the processing. When the end-user application prepares the request to be executed, callback functions can be defined for nominal processing as well as error case processing. If the callback function was set to NULL when the request was prepared then no callback function will be executed. These routines will be executed as part of the device driver so any constraints placed on the device driver will also be placed on the callback routines. So for example, in Linux, copy_to and copy_from user space functions will need to be called.

6.3 Conditional Compilation

The majority of an application will be the same regardless of which operating systems is being used. Some things like semaphores and cache coherency will differ. For these specific differences, conditional compilations are ideal. Code isolation is also a good method of handling porting issues when used in conjunction with conditional compilation. The sample code presented in this document used the following `#defines` to distinguish between VxWorks, Linux kernel, and Linux user applications. In addition, if VXWORKS is not defined, then this code assumes that Linux is being used.

`#define VXWORKS`	for VxWorks applications
`#define __KERNEL__`	for Linux kernel applications (that is, drivers)
`#define _LINUX_USER`	for Linux user applications

In addition, one of the three listed defines must be specified to identify which co-processor is to be used.

MPC185 security processor 60x bus—use `#define MPC185SP`

MPC184 security processor PCI bus—use `#define MPC184SP_pci`

MPC184 security processor 8xx bus—use `#define MPC184SP_8xx`

NOTE

> If more than one of these are defined at the same time, the drivers will not perform the required functions.

6.4 Required Externals

The software driver requires two external variables:

```
// Specifies the base address of the MPC185 security processor in memory.
extern const UINT32 MPC185_BASE_ADRS;
// Specifies the base address of the MPC184 in memory.
extern const UINT32 MPC184_BASE_ADRS;
```

7 VxWorks Environment

The following sections describe the installation of the MPC184 and MPC185 security processor software drivers, BSP integration, and distribution archives.

7.1 Introduction

These release notes support MPC184 and MPC185 security processor software drivers interface for use with VxWorks.

NOTE

Forward slashes are used as pathname delimiters for both UNIX and Windows filenames since this is the default for VxWorks.

7.2 Installation

To install the software drivers, extract the zip file containing the source files (filename.zip) into the Tornado installation directory.

Once the modules are installed, the VxWorks image may be built per the following instructions.

7.3 Building the Interface Modules

Throughout the remainder of the installation instructions, the variables provided in Table 48 are used:

Table 48. VxWorks Interface Module Variables

Variable	Definition
CpuFamily	Specifies the target CPU family, such as **PPCEC603** or **PPC860**
ToolChain	Specifies the tools, such as **gnu**
SecurityProcessor	Specifies the target security processor, such as **MPC184SP_8xx**, **MPC184SP_pci**, or **MPC185SP**

The following steps are used to build drivers:

1. Go to the command prompt or shell
2. Execute **torVars** to set up the Tornado command line build environment.
3. Run **make** in the installDir/**target**/**src**/**drv**/**crypto** directory by typing these command lines switches:

   make CPU=cpuFamily TOOL=toolChain SP=securityProcessor

   (example: make CPU=PPC860 TOOL=gnu SP=MPC184SP_8xx)

The following steps are used to build test code:

1. Go to the command prompt or shell
2. Execute **torVars** to set up the Tornado command line build environment.
3. Run **make** in the installDir/**target**/**src**/**drv**/**crypto** directory by typing these command lines switches followed by testAesa to build the testAesa.c file:

   make CPU=cpuFamily TOOL=toolChain SP=securityProcessor test

   (example: make CPU=PPC860 TOOL=gnu SP=MPC184SP_8xx testAesa)

7.4 BSP Integration

Once the modules are built, they should be linked directly with the user’s board support package, to become integral part of the board image.

In VxWorks, the file **sysLib.c** contains the initialization functions, the memory/address space functions, and the bus interrupt functions. It is recommended to call the function **MPC18xDriverInit** directly from **sysLib.c**.

The security processor will be initialized at board start-up, with all the other devices present on the board.

The same technique can be used in other operating systems.

7.5 Distribution Archive

For this release, the distribution archive consists of the source files listed in this section. Note that the file paths are relative to installDir/target/.

h/drv/crypto/dpd_Table.h	—Data packet descriptor table
h/drv/crypto/MPC18x.h	—User request defines and structures
h/drv/crypto/MPC18x_Descriptors.h	
h/drv/crypto/MPC18xDriver.h	
h/drv/crypto/MPC18xNotify.h	
src/drv/crypto/Makefile	
src/drv/crypto/cha.c	
src/drv/crypto/dpd.c	
src/drv/crypto/init.c	
src/drv/crypto/io.c	
src/drv/crypto/ioctl.c	
src/drv/crypto/isr.c	—interrupt service routine
src/drv/crypto/request.c	

8 Linux Environment

Before starting with the drivers installation, refer to the Readme file included in the Linux package to understand kernel building and board dependencies.

8.1 Installation

To install the software drivers, copy the source files into the build directory of choice.

8.2 Building the Interface Modules

Throughout the remainder of the installation instructions, the variable provided in Table 49 is used:

Table 49. Linus Interface Module Variable

Variable	Definition
SecurityProcessor	Specifies the target security processor, such as **MPC184SP_8xx** [1], **MPC184SP_pci**, or **MPC185SP**

[1] If SP is not specified, MPC184SP_8xx will be the default value.

The following step is used to build drivers and test programs:

1. Run **make** in the /**crypto** directory by typing these command lines:

    make SP=securityProcessor

    (example: make SP=MPC184SP_8xx)

All of the components, which are the driver, the library, and the test application (testDrv.o), will be built.

In addition there is a makefile called des.mk for the user mode program which creates an executable named testDes.

8.3 Integration

Once the modules are built, they may be integrated into the Linux kernel.

8.4 Distribution Archive

For this release, the distribution archive consists of the source files listed in this section. These are the same files as VxWorks plus four additional files, pciLinux.h, RTLinux, mpc18xisr.c, and pciLinux.h.

/board/pci.h	—pci service routines defines
/board/sysPci.c	—pci service routines
/crypto/RTLinux.h	—Linux—VxWorks.h replacement
/crypto/mpc18xisr.c	—Linux—interrupt service routine
/crypto/makefile	

h/drv/crypto/dpd_Table.h	—data packet descriptor table
h/drv/crypto/MPC18x.h	—User request defines and structures
h/drv/crypto/MPC18x_Descriptors.h	
h/drv/crypto/MPC18xDriver.h	
h/drv/crypto/MPC18xNotify.h	
src/drv/crypto/cha.c	
src/drv/crypto/dpd.c	
src/drv/crypto/init.c	
src/drv/crypto/io.c	
src/drv/crypto/ioctl.c	
src/drv/crypto/isr.c	
src/drv/crypto/request.c	

9 Revision History

Table 50 provides a revision history for this user’s manual.

Table 50. Document Revision History

Rev. No.	Substantive Change(s)
0	Initial release.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution
P.O. Box 5405, Denver, Colorado 80217
1-480-768-2130
(800) 521-6274

JAPAN:

Motorola Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu Minato-ku
Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre, 2 Dai King Street
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

(800) 521-6274

HOME PAGE:

www.motorola.com/semiconductors

Information in this document is provided solely to enable system and software implementers to use Motorola products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

MOTOROLA