



MEC2016 ROM API's	
MEC2016 Rom B0 RevB API Reference Manual	
Rev 2.19	08/09/2017

MICROCHIP CONFIDENTIAL

Copyright © 2013 Microchip or its subsidiaries. All rights reserved.

The information contained herein is confidential and proprietary to Microchip, shall be used solely in accordance with the agreement pursuant to which it is provided, and shall not be reproduced or disclosed to others without the prior written consent of Microchip. Although the information is believed to be accurate, no responsibility is assumed for inaccuracies. Microchip reserves the right to make changes to this document and to specifications and product descriptions at any time without notice. Neither the provision of this information nor the sale of the described semiconductor devices conveys any licenses under any patent rights or other intellectual property rights of Microchip or others. The product may contain design defects or errors known as anomalies, including but not necessarily limited to any which may be identified in this document, which may cause the product to deviate from published specifications. Microchip products are not designed, intended, authorized or warranted for use in any life support or other application where product failure could cause or contribute to personal injury or severe property damage. Any and all such uses without prior written approval of an officer of Microchip will be fully at the risk of the customer.

MICROCHIP DISCLAIMS AND EXCLUDES ANY AND ALL WARRANTIES, INCLUDING WITHOUT LIMITATION ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND AGAINST INFRINGEMENT AND THE LIKE, AND ANY AND ALL WARRANTIES ARISING FROM ANY COURSE OF DEALING OR USAGE OF TRADE. IN NO EVENT SHALL SMSC BE LIABLE FOR ANY DIRECT, INCIDENTAL, INDIRECT, SPECIAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES; OR FOR LOST DATA, PROFITS, SAVINGS OR REVENUES OF ANY KIND; REGARDLESS OF THE FORM OF ACTION, WHETHER BASED ON CONTRACT; TORT; NEGLIGENCE OF SMSC OR OTHERS; STRICT LIABILITY; BREACH OF WARRANTY; OR OTHERWISE; WHETHER OR NOT ANY REMEDY OF BUYER IS HELD TO HAVE FAILED OF ITS ESSENTIAL PURPOSE, AND WHETHER OR NOT SMSC HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TABLE OF CONTENTS

1	INTRODUCTION	4
1.1	PURPOSE	4
1.2	SCOPE	4
1.3	REFERENCES	4
1.4	GLOSSARY OF TERMS AND ACRONYMS	4
2	OVERALL DESCRIPTION	5
2.1	PRODUCT PERSPECTIVE	5
2.2	PRODUCT FUNCTIONS	5
2.3	USER CLASSES AND CHARACTERISTICS	5
2.4	DESIGN AND IMPLEMENTATION CONSTRAINTS	5
2.5	ASSUMPTIONS AND DEPENDENCIES	5
3	EXTERNAL INTERFACE REQUIREMENTS	6
3.1	USER INTERFACES	6
3.2	HARDWARE INTERFACES	6
3.3	SOFTWARE INTERFACES	6
4	USAGE	7
4.1	AES FUNCTIONS	7
4.1.1	<i>Powering the Block On/Off</i>	<i>7</i>
4.1.2	<i>Reset the Block</i>	<i>7</i>
4.1.3	<i>Check if the Block is Busy</i>	<i>7</i>
4.1.4	<i>Check the Status of the Block</i>	<i>8</i>
4.1.5	<i>Check the Done Status</i>	<i>8</i>
4.1.6	<i>Stop the Block</i>	<i>9</i>
4.1.7	<i>Start the Block</i>	<i>9</i>
4.1.8	<i>Clear the Interrupts</i>	<i>9</i>
4.1.9	<i>Set the Private Key and Initialization vector</i>	<i>10</i>
4.1.10	<i>Encrypt/Decrypt</i>	<i>12</i>
4.2	RSA FUNCTIONS	13
4.2.1	<i>Load RSA Keys into the Crypto Memory</i>	<i>13</i>
4.2.2	<i>Load CRT Parameters into the Crypto Memory</i>	<i>18</i>
4.2.3	<i>Read from the Crypto Memory</i>	<i>19</i>
4.2.4	<i>Perform RSA Operations</i>	<i>22</i>
4.2.5	<i>RSA Signature Verification</i>	<i>23</i>
4.2.6	<i>RSA Encrypt/Decrypt</i>	<i>24</i>
4.3	PKE FUNCTIONS	25
4.3.1	<i>Powering the Block On/Off</i>	<i>25</i>
4.3.2	<i>Reset the Block</i>	<i>26</i>
4.3.3	<i>Check the Status of the Block</i>	<i>26</i>
4.3.4	<i>Check the Done Status</i>	<i>27</i>
4.3.5	<i>Start the Block</i>	<i>27</i>
4.3.6	<i>Check if the Block is Busy</i>	<i>27</i>
4.3.7	<i>Set Operand Slot(s)</i>	<i>28</i>
4.3.8	<i>Get Slot Details</i>	<i>29</i>
4.3.9	<i>Clear Slots in the Crypto Memory</i>	<i>29</i>
4.3.10	<i>Read from the Crypto Memory</i>	<i>30</i>
4.3.11	<i>Write to the Crypto Memory</i>	<i>31</i>
4.3.12	<i>Read and Clear PKE Status</i>	<i>32</i>
4.3.13	<i>Perform Modular Arithmetic</i>	<i>33</i>
4.4	ELLIPTIC CURVE FUNCTIONS	34
4.4.1	<i>Elliptic Curve Arithmetic Operations</i>	<i>34</i>
4.4.2	<i>Checking Elliptic Curve Parameters</i>	<i>37</i>
4.4.3	<i>Load Elliptic Curve Parameters into the Crypto Memory</i>	<i>40</i>

4.4.4	<i>ECDSA Signature Verification</i>	42
4.4.5	<i>KCDSA Functions</i>	43
4.4.6	<i>SRP Operation</i>	45
4.4.7	<i>Elliptic Curve 25519 Point Multiplication</i>	46
4.4.8	<i>Edward's Curve 25519 Operations</i>	47
4.5	RNG FUNCTIONS	50
4.5.1	<i>Powering the Block On/Off</i>	50
4.5.2	<i>Reset the Block</i>	50
4.5.3	<i>Set the Mode of Operation</i>	50
4.5.4	<i>Check if the Block has Started</i>	51
4.5.5	<i>Start the block</i>	51
4.5.6	<i>Stop the block</i>	51
4.5.7	<i>Get Data from FIFO</i>	52
4.5.8	<i>Retrieve Data as Bytes from FIFO</i>	52
4.5.9	<i>Retrieve Data as Words from FIFO</i>	52
4.6	HASH FUNCTIONS	53
4.6.1	<i>Check the Status of the Block</i>	53
4.6.2	<i>Check if the Block is Busy</i>	53
4.6.3	<i>Starting the Hash Block</i>	54
4.6.4	<i>Check the Done Status</i>	54
4.6.5	<i>Clear Hash Interrupts</i>	55
4.7	SHA FUNCTIONS	56
4.7.1	<i>SHA Direct Functions</i>	56
4.7.2	<i>SHA1 Functions</i>	58
4.7.3	<i>SHA256 Functions</i>	61
4.7.4	<i>SHA512 Functions</i>	65
4.8	MISCELLANEOUS ROM API	67
4.8.1	<i>Check Version Number of the APIs</i>	67
4.8.2	<i>Load Firmware</i>	68
5	APPENDIX:	70
6	REVISION HISTORY	73

1 Introduction

1.1 Purpose

The document illustrates the usage of ROM API's available in MEC 2016

1.2 Scope

This document will serve as a usage manual for the functions provided by the function in the ROM. It presents the reader with the function header, a description of the functions operations, it inputs and output parameters.

1.3 References

-

1.4 Glossary of Terms and Acronyms

SPI – Serial Peripheral Interface
AES – Advanced encryption Standard
RSA - Rivest-Shamir-Adleman cryptosystem
PKE – Public Key Encryption
SHA – Secure Hash Algorithm
RNG – Random Number Generator
SCM – Shared Crypto Memory
CRT – Chinese Remainder Theorem
KCDSA – Korean Elliptic Curve Digital Signature Algorithm
ECDSA – Elliptic Curve Digital Signature Algorithm
EC25519 – Elliptic Curve 25519
EC – Elliptic Curve
SRP – Secure Remote Password
DMA – Direct Memory Access

2 Overall Description

2.1 Product Perspective

The Boot Rom API allows software access to certain hardware features that facilitates easy development of application on the MEC2016 platform. These API's serve the function of providing easy access to the underlying hardware features.

2.2 Product Functions

The Boot Rom API's provide software access features like access to SPI/FLASH, AES encryption, RSA Crypt engine, Public Key Encryption. All operations are abstracted by Application program interfaces and the programmer need only to use the API's to leverage these device specific operations.

2.3 User Classes and Characteristics

This document is intended for programmers and users of the MEC2016 product. It illustrates the use of certain software features that would facilitate easy development of applications.

2.4 Design and Implementation Constraints

The API's are all ROM resident. They only use stack dynamic variables and internal reference/pointers (these constraints discount any pointers passed by the user to the functions). The API's do not use heap dynamic or global space to store data.

2.5 Assumptions and Dependencies

The efficacy of this user manual may be contingent upon the knowledge of the MEC2016 target hardware. Certain references in this document may require the usage of device data sheets.

3 External Interface Requirements

3.1 User Interfaces

The document will describe the user interface to the rom resident API's.

3.2 Hardware Interfaces

MEC2016 EVB / FPGA with proper bit map
Keil µVision Ulink Pro Debugger tool –
MCHP Trace debugger Tool
Dediprog SPI programmer

3.3 Software Interfaces

Keil Compiler IDE-Version:
µVision V5.15

Tool Version Numbers:

Toolchain:	MDK-ARM Standard Cortex-M Version: 5.15.0	
Toolchain Path:	C:\Keil_v5\ARM\ARMCC\Bin	
C Compiler:	Armcc.exe	V5.05 update 2 (build 169)
Assembler:	Armasm.exe	V5.05 update 2 (build 169)
Linker/Locator:	ArmLink.exe	V5.05 update 2 (build 169)
Library Manager:	ArmAr.exe	V5.05 update 2 (build 169)
Hex Converter:	FromElf.exe	V5.05 update 2 (build 169)
CPU DLL:	SARMCM3.DLL	V5.15.0
Dialog DLL:	DCM.DLL	V1.13.2.0
Target DLL:	ULP2CM3.DLL	V2.200.17.0
Dialog DLL:	TCM.DLL	V1.14.5.0

-

4 Usage

Note1: All blocks need to be powered ON with corresponding APIs before usage of any of the API for crypto operations.

4.1 AES Functions

4.1.1 Powering the Block On/Off

Functions:

api_aes_hash_power

Function Header:

```
Void api_aes_hash_power(uint8_t pwr_state);
```

Description:

This function is used to enable or disable AES and Hash Hardware Block.

Note: AES and Hash hardware accelerators do not implement a block level clock gate control and also share AHB resources (master, internal DMA, clocking). A single PCR sleep control will sleep both blocks. Before setting the PCR sleep bit, clear AES and Hash Control registers to stop both engines. To wake (ungated clocks) clear the PCR sleep enable bit.

Inputs:

Input Parameter	Description
Pwr_state	A uint8_t value which if 0, puts the module to sleep (gate off clocks to block); if 1 enables the block (gate on clocks to block).

Outputs: None

4.1.2 Reset the Block

Functions:

api_aes_hash_reset

Function Header:

```
Void api_aes_hash_reset(void);
```

Description:

This function is used to reset the AES and Hash block.

Inputs: None

Outputs: None

4.1.3 Check if the Block is Busy

Functions:

api_aes_busy

Function Header:

Bool api_aes_busy(void);

Description:

These functions are used to check if the AES block is running. "True" is returned if the block is running and "false" is returned if the block is not running.

Inputs: None

Outputs:

True if AES block is running, False otherwise.

4.1.4 Check the Status of the Block

Functions:

api_aes_status

Function Header:

uint32_t api_aes_status(void);

Description:

This function is used to read the status of the AES block.

Inputs: none

Outputs:

The status of the AES block is reflected in the return value. The bit definitions of AES Status value is presented below.

Bit Number	Definition (all bits are read only)
0	AES Busy Status (1 if busy, 0 otherwise)
1	AES CCM Mode MAC_T Calculation is valid (1 if valid, 0 otherwise; <i>Not Supported</i>)

4.1.5 Check the Done Status

Functions:

api_aes_is_done_status2

Function Header:

Bool api_aes_is_done_status2(uint32_t* status_value);

Description:

The function updates the status value of the AES block in the pointer argument. The done status is evaluated and it is returned as a Boolean value.

Inputs:

Input Parameter	Description
Status_value	An unsigned 32 bit integer pointer where the status value will be stored.

Outputs:

The status of the AES is updated in the pointer argument. If TRUE is returned, it means that the AES operation is done, FALSE otherwise. The bit definitions of AES Status values are:

Bit Number	Definition (all bits are read only)
0	AES Busy Status (1 if busy, 0 otherwise)
1	AES CCM Mode MAC_T Calculation is valid (1 if valid, 0 otherwise; <i>Not Supported</i>)

4.1.6 Stop the Block

Functions:

api_aes_stop

Function Header:

```
Bool api_aes_stop( void )
```

Description:

This function is used to stop the AES block. The function accesses the AES control register to stop the AES Operations.

Inputs: None

Outputs:

This function returns the busy status of AES block as a Boolean value. "True" is returned if the block is running and "false" is returned if the block is not running.

4.1.7 Start the Block

Functions:

api_aes_start

Function Header:

```
void api_aes_start(uint8_t ien);
```

Description:

This function is used to start the AES Block. The input argument controls interrupt enable

Inputs:

Input Parameter	Description
ien	Uint8_t value specifying the interrupt status. This value, if non-zero enables interrupt and if 0, disables interrupt.

Outputs:

None

4.1.8 Clear the Interrupts

Functions:

api_aes_iclr

Function Header:

```
Uint32_t api_aes_iclr(void);
```

Description:

The function is used to clear AES interrupts. AES Status register is a read-to-clear register.

Inputs: NoneOutputs:

This function returns 0 if the operation was successful; 1 otherwise.

4.1.9 Set the Private Key and Initialization vector

4.1.9.1 api_aes_set_key

Function Header:

```
Uint8_t api_aes_set_key ( const uint32_t *pkey,
                          const uint32_t *piv,
                          uint8_t key_len,
                          bool msbf)
```

Description:

The function programs the AES accelerator with key and optional initialization. AES key size and pre calculation mode are set. The AES engine is not started. Do not call this routine if the AES engine is busy.

Inputs:

Input parameter	Description
Pkey	An unsigned 32 bit pointer pointing to an aligned buffer containing the AES key, LSB first
Piv	An unsigned 32 bit pointer pointing to an aligned buffer containing AES initialization vector, LSB first. NULL if no Initialization vector is required.
Key_len	A uint8_t indicating the key length. The permitted values are <ul style="list-style-type: none"> • AES_KEYLEN_128 • AES_KEYLEN_192 • AES_KEYLEN_256.
Msbf	A Boolean value indicating most significant bit first. If AES key and initialization vector are in most significant bit first order, it should be true; otherwise false.

Macro values for key length:

Macro Name	Value
AES_KEYLEN_128	0
AES_KEYLEN_192	1
AES_KEYLEN_256	2

Outputs:

AES_OK (Success), AES_ERR_BAD_POINTER (pkey is NULL), AES_ERR_BAD_KEY_LEN (key len is not supported).

Macro values for return codes:

Macro Name	Value
AES_OK	0
AES_ERR_BAD_KEY_LEN	2
AES_ERR_BAD_POINTER	3

4.1.9.2 api_aes_prog_key

Function Header:

```
uint8_t api_aes_prog_key( const uint8_t *pkey,
                        uint8_t key_len,
                        uint8_t msbf);
```

Description:

The function programs the AES accelerator with the key. AES key size and pre calculation mode are set.

Inputs:

Input parameter	Description
Pkey	An unsigned 8 bit pointer pointing to an aligned buffer containing the AES key, LSB first
Key_len	A uint8_t indicating the key length. The permitted values are <ul style="list-style-type: none"> • AES_KEYLEN_128 • AES_KEYLEN_192 • AES_KEYLEN_256.
Msbf	A uint8_t value indicating most significant bit first. If the AES key is in most significant bit first order, it should be 1; otherwise 0.

Macro values for key length:

Macro Name	Value
AES_KEYLEN_128	0
AES_KEYLEN_192	1
AES_KEYLEN_256	2

Outputs:

AES_OK (Success), AES_ERR_BAD_POINTER (pkey is NULL), AES_ERR_BAD_KEY_LEN (key len is not supported).

Macro values for return codes:

Macro Name	Value
AES_OK	0
AES_ERR_BAD_KEY_LEN	2
AES_ERR_BAD_POINTER	3

4.1.9.3 api_aes_prog_iv

Function Header:

```
uint8_t api_aes_prog_iv(const uint8_t *piv,
                        uint8_t iv_len,
                        uint8_t msbf);
```

Description:

The function programs the AES accelerator with the initialization.

Inputs:

Input parameter	Description
Piv	An unsigned 8 bit pointer pointing to an aligned buffer containing AES initialization vector, LSB first.
iv_len	A uint8_t value indicating the length of the initialization.
Msbf	A uint8_t value indicating most significant bit first. If the initialization vector is in most significant bit first order, it should be 1; otherwise 0.

Outputs:

AES_OK (Success), AES_ERR_BAD_POINTER (pkey is NULL), AES_ERR_BAD_KEY_LEN (key len is not supported).

Macro values for return codes:

Macro Name	Value
AES_OK	0
AES_ERR_BAD_KEY_LEN	2
AES_ERR_BAD_POINTER	3

4.1.10 Encrypt/Decrypt

Functions:

api_aes_crypt

Function Header:

```
uint8_t api_aes_crypt (const uint32_t *data_in,
                      uint32_t *data_out,
                      uint32_t num_128bit_blocks,
                      uint8_t modes);
```

Description:

This function programs specified AES operation using currently programmed key. The hardware permits the following modes of operation; ECB, CBC, CTR and OFB modes.

Inputs:

Input Parameters	Description
------------------	-------------

Data_in	A pointer to a word (32-bit) aligned input data buffer.
Data_out	A pointer to a word (32-bit) aligned output data buffer.
Num_128bit_blocks	An unsigned 32 bit value specifying the size of input data as a number of 128-bit (16-byte) blocks.
Modes	An unsigned 8 bit integer indicating encryption/decryption the mode of operation. The permitted modes of operation for encryption are <ul style="list-style-type: none"> • AES_MODE_ECB • AES_MODE_CBC • AES_MODE_CTR • AES_MODE_OFB.

Macro values for Modes field:

Macro Name	Value
AES_MODE_ECB	0
AES_MODE_CBC	1
AES_MODE_CTR	2
AES_MODE_CFB	3
AES_MODE_OFB	4
AES_MODE_DECRYPT	0x80

Outputs:

AES_OK (AES HW Programmed), AES_ERR_BAD_POINTER (NULL pointers or number of blocks is 0).

Macro values for return codes:

Macro Name	Value
AES_OK	0
AES_ERR_BAD_POINTER	3

4.2 RSA Functions

Note: Before using RSA functions the PKE block needs to be powered ON explicitly using `api_pke_power()`.

4.2.1 Load RSA Keys into the Crypto Memory

4.2.1.1 `api_pke_rsa_load_param`

Function Header:

```
uint8_t api_pke_rsa_load_param (uint16_t param_loc,
                                const uint8_t *p,
                                uint16_t nbytes,
                                uint8_t msbf);
```

Description:

This function loads the RSA parameters into the specified slot in the SCM.

Inputs:

Input parameters	Description
Param_loc	An unsigned 16 bit integer specifying the slot number the parameter is to be loaded into.
p	An unsigned 8 bit integer pointer to data that is to be written into the location
nbytes	An unsigned 16 bit integer indicating the number of bytes to be written into the memory
Msbfb	An unsigned 8 bit integer which if 1, indicates Most significant byte first and if 0, indicates least significant byte first.

Outputs:

The return values and their description is presented below.

PKE_RET_ERR_BUSY – if the RSA module is busy

PKE_RET_ERR_BAD_ADDR– if the pointer to the data that is to be written is NULL.

PKE_RET_OK – if the operation requested was successful.

PKE_RET_ERR_BAD_PARAM – if 0 bytes were copied into the SCM location

Return code macro values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BUSY	1
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_BAD_ADDR	3

4.2.1.2 api_pke_rsa_load_pub_key

Function Header:

```
uint8_t api_pke_rsa_load_pub_key (uint16_t rsa_key_bitlen,
                                   const uint8_t *pmod,
                                   const uint8_t *pexp,
                                   uint16_t explen,
                                   uint8_t msbf);
```

Description:

This function loads the RSA public key into the SCM. If moduli pointers are NULL, no copy operations are performed.

Inputs:

Input parameters	Description
rsa_key_bitlen	An unsigned 16 bit integer reflecting the size of the RSA key. The permitted values are 1024, 2048 or 4096 bits.
pmod	An unsigned 8 bit integer pointer to the RSA Public Modulus.
pexp	An unsigned 8 bit integer pointer to the RSA Public Exponent.
explen	An unsigned 16 bit integer specifying the number of bytes in the Public Exponent

msbf	An unsigned 8 bit integer which if 1, indicates Most significant byte first and if 0, indicates least significant byte first.
------	---

Outputs:

The return values and their description is presented below.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if rsa_key_bitlen has invalid data.

PKE_RET_OK – if the operation requested was successful.

PKE_RET_ERR_BAD_PARAM – if 0 bytes were copied into the SCM location.

Return code macro values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_INVALID_BIT_LENGTH	5

4.2.1.3 api_pke_rsa_load_prv_key

Function Header:

```
uint8_t api_pke_rsa_load_prv_key (uint16_t rsa_key_bitlen,
                                   const uint8_t *pmod,
                                   const uint8_t *prvexp,
                                   uint8_t msbf);
```

Description:

This function loads RSA public modulus and private exponent into the SCM. If moduli pointers are NULL, no copy operations are performed.

Inputs:

Input parameters	Description
rsa_key_bitlen	An unsigned 16 bit integer reflecting the size of the RSA key. The permitted values are 1024, 2048 or 4096 bits.
pmod	An unsigned 8 bit integer pointer to the RSA Public Modulus.
prvexp	An unsigned 8 bit integer pointer to the RSA Private Exponent.
msbf	An unsigned 8 bit integer which if 1, indicates Most significant byte first and if 0, indicates least significant byte first.

Outputs:

The return values and their descriptions are presented below.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if rsa_key_bitlen has invalid data.

PKE_RET_OK – if the operation requested was successful.

PKE_RET_ERR_BAD_PARAM – if 0 bytes were copied into the SCM location.

Return code macro values:

Macro Name	Value
------------	-------

PKE_RET_OK	0
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_INVALID_BIT_LENGTH	5

4.2.1.4 api_pke_rsa_load_pub_keystruct

Function Header:

```
uint8_t api_pke_rsa_load_pub_keystruct (const RSA_PUB_KEY *pkey);
```

Description:

This function loads the public key into the PKE shared crypto memory. If moduli pointers are NULL, no copy operations are performed.

Inputs:

Input parameters	Description
pkey	A pointer to the structure RSA_PUB_KEY defined below.

Structure definition of RSA_PUB_KEY

```
typedef struct rsa_pub_key RSA_PUB_KEY;
struct rsa_pub_key {
    uint32_t key_bitlen;
    PKE_BIG_INT mod;
    PKE_BIG_INT exp;
};
```

where, key_bitlen – An unsigned 32 bit integer specifying the length of the RSA key. Permitted values are 1024 bits, 2048 bits or 4096 bits.

mod - PKE_BIG_INT structure containing details about the RSA Public Modulus

exp - PKE_BIG_INT structure containing details about the RSA Public Exponent

Structure definition of PKE_BIG_INT

```
typedef struct pke_big_int PKE_BIG_INT;
struct pke_big_int
{
    uint16_t flags;
    uint16_t blen;
    uint8_t *pbytes;
};
```

where, flags – An unsigned 16 bit integer specifying if MSBF. If bit[0]=0/1 (LSBF/MSBF); if bit[7]=0/1 (positive/negative).

blen – An unsigned 16 bit integer specifying the length of the key/exponent in bytes

pbytes – An unsigned 8 bit pointer to the key/exponent.

Outputs:

The return values and their descriptions are presented below.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if the size of the key is invalid.

PKE_RET_OK – if the operation requested was successful.

PKE_RET_ERR_BAD_PARAM – if pkey has the value NULL or if 0 bytes were copied into the SCM location.

Return code macro values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_INVALID_BIT_LENGTH	5

4.2.1.5 api_pke_rsa_load_key

Function Header:

```
uint8_t api_pke_rsa_load_key (const RSA_KEY *pkey);
```

Description:

This function loads RSA keys into the PKE shared crypto memory. If moduli pointers are NULL, no copy operations are performed.

Inputs:

Input parameters	Description
pkey	A pointer to the structure RSA_KEY defined below.

Structure definition of RSA_KEY

```
typedef struct rsa_key RSA_KEY;
struct rsa_key {
    RSA_PUB_KEY pub_key;
    PKE_BIG_INT prv_exp;
};
```

where, pub_key – RSA_PUB_KEY structure containing details about RSA Public Key
 prv_exp – PKE_BIG_INT structure containing details about RSA Private Exponent

Structure definition of RSA_PUB_KEY

```
typedef struct rsa_pub_key RSA_PUB_KEY;
struct rsa_pub_key {
    uint32_t key_bitlen;
    PKE_BIG_INT mod;
    PKE_BIG_INT exp;
};
```

where, key_bitlen – An unsigned 32 bit integer specifying the length of the RSA key. Permitted values are 1024 bits, 2048 bits or 4096 bits.
 mod - PKE_BIG_INT structure containing details about the RSA Public Modulus
 exp - PKE_BIG_INT structure containing details about the RSA Public Exponent

Structure definition of PKE_BIG_INT

```
typedef struct pke_big_int PKE_BIG_INT;
struct pke_big_int
{
    uint16_t flags;
    uint16_t blen;
    uint8_t *pbytes;
};
```

where, flags – An unsigned 16 bit integer specifying if MSBF. If bit[0]=0/1 (LSBF/MSBF); if bit[7]=0/1 (positive/negative).
 blen – An unsigned 16 bit integer specifying the length of the key/exponent in bytes
 pbytes – An unsigned 8 bit pointer to the key/exponent.

Outputs:

The return values and their descriptions are presented below.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if the size of the key is invalid.

PKE_RET_OK – if the operation requested was successful.

PKE_RET_ERR_BAD_PARAM – if pkey has the value NULL or if 0 bytes were copied into the SCM location.

Return code macro values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_INVALID_BIT_LENGTH	5

4.2.2 Load CRT Parameters into the Crypto Memory

Function:

api_pke_rsa_load_crt_key

Function Header:

uint8_t api_pke_rsa_load_crt_key (const RSA_CRT_KEY *pkey);

Description:

This routine loads CRT Key parameters into their respective slots in the SCM.

Inputs:

Input parameters	Description
pkey	A pointer to the structure RSA_CRT_KEY defined below.

Structure definition of RSA_CRT_KEY

```
typedef struct rsa_crt_key RSA_CRT_KEY;
struct rsa_crt_key {
    uint32_t key_bitlen;
    PKE_BIG_INT prime1;
    PKE_BIG_INT prime2;
    PKE_BIG_INT crt_dp;
    PKE_BIG_INT crt_dq;
```

```
PKE_BIG_INT crt_i;
};
```

where, key_bitlen – An unsigned 32 bit integer specifying the length of the RSA key. Permitted values are 1024 bits, 2048 bits or 4096 bits.

Prime1 - PKE_BIG_INT structure containing details about prime integer 1

Prime2 - PKE_BIG_INT structure containing details about prime integer 2

crt_dp - PKE_BIG_INT structure containing details about exponent 1

crt_dq - PKE_BIG_INT structure containing details about exponent 2

crt_i - PKE_BIG_INT structure containing details about the coefficient

Structure definition of PKE_BIG_INT

```
typedef struct pke_big_int PKE_BIG_INT;
struct pke_big_int
{
    uint16_t flags;
    uint16_t blen;
    uint8_t *pbytes;
};
```

where, flags – An unsigned 16 bit integer specifying if MSBF. If bit[0]=0/1 (LSBF/MSBF); if bit[7]=0/1 (positive/negative).

blen – An unsigned 16 bit integer specifying the length of the data in bytes

pbytes – An unsigned 8 bit pointer to the data.

Outputs:

The return values and descriptions are presented below.

PKE_RET_OK if successful

PKE_RET_ERR_BUSY if pke is busy

PKE_RET_ERR_INVALID_BIT_LENGTH if key_bitlen has an invalid value.

PKE_RET_ERR_BAD_ADDR– if the pointer to the data that is to be written is NULL.

PKE_RET_ERR_BAD_PARAM – if 0 bytes were copied into the SCM location

Return code macro values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BUSY	1
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_BAD_ADDR	3
PKE_RET_ERR_INVALID_BIT_LENGTH	5

4.2.3 Read from the Crypto Memory

4.2.3.1 api_pke_rsa_get_param

Function Header:

```
uint8_t api_pke_rsa_get_param (uint16_t param_loc,
                               uint8_t *p,
                               uint16_t nbytes,
                               uint8_t msbf)
```

Description:

This function reads the parameters from the specified slot number.

Inputs:

Input Parameters	Description
Param_loc	An unsigned 16 bit integer specifying the slot number to be read.
p	An unsigned 8 bit integer pointer to location the data is to be written into.
nbytes	An unsigned 16 bit integer indicating the number of bytes to be read.
Msbf	An unsigned 8 bit integer which if 1, indicates Most significant byte first and if 0, indicates least significant byte first.

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if the pointer to the location points to NULL.

PKE_RET_ERR_BUSY – This error value is returned if the pke block is busy.

PKE_RET_ERR_BAD_PARAM – this error value is returned if 0 bytes were read.

PKE_RET_OK – this value indicates a successful execution of requested operation.

Return code macro values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BUSY	1
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_BAD_ADDR	3

4.2.3.2 api_pke_rsa_get_key

Function Header:

```
uint8_t api_pke_rsa_get_key (RSA_KEY *pkey,
                             uint32_t pubexp_len,
                             uint8_t msbf);
```

Description:

This function reads the RSA keys from the SCM. If moduli pointers are NULL, no copy operations are performed.

Inputs:

Input Parameters	Description
pkey	A pointer to the structure RSA_KEY defined below.
pubexp_len	An unsigned 32 bit integer specifying the length of the public exponent in bytes.
Msbf	An unsigned 8 bit integer which if 1, indicates Most significant byte first and if 0,

	indicates least significant byte first.
--	---

Structure definition of RSA_KEY

```
typedef struct rsa_key RSA_KEY;
struct rsa_key {
    RSA_PUB_KEY pub_key;
    PKE_BIG_INT prv_exp;
};
```

where, pub_key – RSA_PUB_KEY structure containing details about RSA Public Key
 prv_exp – PKE_BIG_INT structure containing details about RSA Private Exponent

Structure definition of RSA_PUB_KEY

```
typedef struct rsa_pub_key RSA_PUB_KEY;
struct rsa_pub_key {
    uint32_t key_bitlen;
    PKE_BIG_INT mod;
    PKE_BIG_INT exp;
};
```

where, key_bitlen – An unsigned 32 bit integer specifying the length of the RSA key. Permitted values are 1024 bits, 2048 bits or 4096 bits.
 mod - PKE_BIG_INT structure containing details about the RSA Public Modulus
 exp - PKE_BIG_INT structure containing details about the RSA Public Exponent

Structure definition of PKE_BIG_INT

```
typedef struct pke_big_int PKE_BIG_INT;
struct pke_big_int
{
    uint16_t flags;
    uint16_t blen;
    uint8_t *pbytes;
};
```

where, flags – An unsigned 16 bit integer specifying if MSBF. If bit[0]=0/1 (LSBF/MSBF); if bit[7]=0/1 (positive/negative).
 blen – An unsigned 16 bit integer specifying the length of the key/exponent in bytes
 pbytes – An unsigned 8 bit pointer to the key/exponent.

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if the pointer points to NULL.

PKE_RET_ERR_BUSY – This error value is returned if the pke block is busy.

PKE_RET_ERR_BAD_PARAM – this error value is returned if 0 bytes were read.

PKE_RET_OK – this value indicates a successful execution of requested operation.

Return code macro values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BUSY	1

PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_BAD_ADDR	3

4.2.4 Perform RSA Operations

Function:

api_pke_rsa_operation

Function Header:

```
uint8_t api_pke_rsa_operation (uint16_t key_bitlen,
                               uint32_t op,
                               uint32_t flags);
```

Description:

This function performs the following RSA operations:

- RSA_DO_MOD_EXP
- RSA_DO_KEYGEN
- RSA_DO_CRT_PGEN
- RSA_DO_CRT_DECRYPT
- RSA_DO_ENCRYPT
- RSA_DO_DECRYPT
- RSA_DO_SIG_GEN
- RSA_DO_VERIFY

All the keys/parameters need to be loaded into their respective slots before calling this function.

Macro value	Operation	Input Parameters	slot number	Output Parameters
6	RSA_DO_SIG_GEN	Public Modulus (n)	Slot 0	RSA Signed Hash (S)
		Private Exponent (d)	Slot 6	
		Public Exponent (e)	Slot 8	
		Hash Digest	Slot 4	
7	RSA_DO_VERIFY	Sign to be verified (S)	Slot 4	Verified Signature Output

The results for all the operations can be read from Slot 5.

Inputs:

Input Parameters	Description
Key_bitlen	An unsigned 16 bit integer specifying the length of the key: 1024, 2048 or 4096
op	An unsigned 32 bit integer indicating the RSA operation to be performed. It can take the following values: <ul style="list-style-type: none"> • RSA_DO_MOD_EXP • RSA_DO_KEYGEN • RSA_DO_CRT_PGEN • RSA_DO_CRT_DECRYPT • RSA_DO_ENCRYPT • RSA_DO_DECRYPT • RSA_DO_SIG_GEN • RSA_DO_VERIFY

flags	An unsigned 32 bit integer specifying the interrupt status. If bit 5 is set, it enables the done and error interrupts.
-------	--

Macro values for op field:

Macro Name	Value
RSA_DO_MOD_EXP	0
RSA_DO_KEYGEN	1
RSA_DO_CRT_PGEN	2
RSA_DO_CRT_DECRYPT	3
RSA_DO_ENCRYPT	4
RSA_DO_DECRYPT	5
RSA_DO_SIG_GEN	6
RSA_DO_VERIFY	7

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_ERR_UNKNOWN_OP – This error value is returned if op is invalid.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if key_bitlen has invalid data.

PKE_RET_OK – this value indicates a successful execution of requested operation.

Return code macro values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BUSY	1
PKE_RET_ERR_UNKNOWN_OP	4
PKE_RET_ERR_INVALID_BIT_LENGTH	5

4.2.5 RSA Signature Verification

Function:

api_pke_rsa_is_signature_valid

Function Header:

```
uint8_t api_pke_rsa_is_signature_valid (void);
```

Description:

This routine is used check if the RSA signature is valid.

Inputs:

None

Outputs:

This function returns 1 if the signature is valid, 0 otherwise.

4.2.6 RSA Encrypt/Decrypt

4.2.6.1 api_pke_rsa_crypt

Function Header:

```
uint8_t api_pke_rsa_crypt (const uint8_t *pdata,
                           uint32_t nbytes,
                           uint16_t flags);
```

Description:

This function performs RSA Encryption/Decryption using modular arithmetic. Before calling this function, the keys need to be loaded into their respective slots.

Inputs:

Input Parameters	Description
pdata	An unsigned 8 bit integer pointer to the message
nbytes	An unsigned 32 bit integer specifying the number of bytes in the message
flags	An unsigned 16 bit integer specifying the following: <ul style="list-style-type: none"> • b[0] = 0/1 (LSBF/MSBF) • b[1] = 0/1 (encrypt or decrypt with public key/ encrypt or decrypt with private key) • b[4] = 0/1 (do not start/start) • b[5] = 0/1 (if start, do not enable interrupts/ if start enable interrupts)

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR– this error value is returned if pdata is NULL

PKE_RET_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_ERR_BAD_PARAM – if 0 bytes were copied into the SCM location

PKE_RET_OK – this value indicates a successful execution of requested operation.

Return code macro values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BUSY	1
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_BAD_ADDR	3

4.2.6.2 api_pke_rsa_crt_decrypt

Note: Set the PKE command register after calling api_pke_rsa_crt_decrypt ():

```
api_pke_rsa_crt_decrypt (&pdata, nbytes, flags);
pke_command &= ~(0x17)
pke_command |= 0x13
```


Function Header:

```
uint8_t api_pke_rsa_crt_decrypt (const uint8_t *pdata,
                                uint32_t nbytes,
                                uint16_t flags);
```

Description:

This routine performs RSA decryption using Chinese remainder theorem. The parameters need to be loaded into the slots before calling this function.

Inputs:

Input parameter	Description
pdata	An unsigned 8 bit integer pointer to the data
nbytes	An unsigned 32 bit integer specifying the length of the data in bytes
flags	An unsigned 16 bit integer specifying the following: <ul style="list-style-type: none"> • b[0] = 0/1 (LSBF/MSBF) • b[1] = HW always performs decryption using private exponent • b[2] = ignored, always perform decryption • b[4] = 0/1 (do not start/start) • b[5] = 0/1 (if start, do not enable interrupts/if start, enable interrupts)

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_BAD_ADDR – if pdata is NULL.

PKE_RET_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation.

PKE_RET_ERR_BAD_PARAM – if 0 bytes were copied into the SCM location

Return code macro values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BUSY	1
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_BAD_ADDR	3

4.3 PKE Functions

4.3.1 Powering the Block On/Off

Functions:

```
api_pke_power
```

Function Header:

```
Void api_pke_power (bool pwr_on);
```

Description:

This routine controls the Gate on/off clocks to pke block. Before setting PCR PKE sleep enable, write 0 to the PKE control register to stop the engine.

Inputs:

Input Parameters	Description
Pwr_on	A Boolean input parameter. True wakes the block (ungates clocks), False puts the block in sleep mode (gates clock)

Outputs:

None

4.3.2 Reset the Block

Functions:

api_pke_reset

Function Header:

Void api_pke_reset (void);

Description:

This routine resets the pke block to the hardware default.

Inputs:

None

Output:

None

4.3.3 Check the Status of the Block

Functions:

api_pke_status

Function Header:

Uint32_t api_pke_status (void);

Description:

The routine returns the status of the PKE block.

Inputs:

None

Outputs:

The routine returns the pke block status. The important status bits are listed below:

Bit Number	Description
16	PKE Busy Bit. This applies to all operations. (1 if busy, 0 otherwise)
10	This bit applies to Elliptic curve, 0 if parameters a,b are valid, 1 otherwise.
9	This bit only applies to signature verify operations. 0 if signature is valid, 1 otherwise.
7	This bit applies to Elliptic curve, 0 if parameter n is valid, 1 otherwise.
6	This bit applies to Elliptic curve, 0 if couple x,y is valid, 1 otherwise.
5	This bit applies to Elliptic Curve Operations. 0 if EC point not at infinity, 1 otherwise
4	This bit applies to Elliptic Curve Operations. 0 if EC point is on curve, 1 otherwise.

4.3.4 Check the Done Status

Functions:

api_pke_is_done_status2

Function Header:

```
Bool api_pke_is_done_status2 (uint32_t *status_value);
```

Description:

The routine is used to check the done status of the pke block.

Inputs:

Inputs Parameters	Description
Status_value	An unsigned 32 bit integer pointer where the status value will be stored.

Outputs:

Returns true if the PKE operation is done, false otherwise. The important status bits are listed below:

Bit Number	Description
16	PKE Busy Bit. This applies to all operations. (1 if busy, 0 otherwise)
10	This bit applies to Elliptic curve, 0 if parameters a,b are valid, 1 otherwise.
9	This bit only applies to signature verify operations. 0 if signature is valid, 1 otherwise.
7	This bit applies to Elliptic curve, 0 if parameter n is valid, 1 otherwise.
6	This bit applies to Elliptic curve, 0 if couple x,y is valid, 1 otherwise.
5	This bit applies to Elliptic Curve Operations. 0 if EC point not at infinity, 1 otherwise
4	This bit applies to Elliptic Curve Operations. 0 if EC point is on curve, 1 otherwise.

4.3.5 Start the Block

Functions:

api_pke_start

Function Header:

```
void api_pke_start(bool ien);
```

Description:

The function starts the pke block.

Inputs:

Input Parameters	Description
ien	A Boolean value specifying the interrupt status. If the interrupt enable mask is set, it enables the done and error interrupts.

Outputs:

None

4.3.6 Check if the Block is Busy

Functions:

Microchip Confidential,

API Manual

api_pke_busy

Function Header:

Bool api_pke_busy(void);

Description:

This routine returns the PKE busy status. PKE status register bits are read-only and are cleared upon reading. Reading also clears PKE block interrupt signal.

Inputs: none

Outputs:

This routine returns a true if PKE is busy, False otherwise.

4.3.7 Set Operand Slot(s)

4.3.7.1 api_pke_set_operand_slot

Function Header:

void api_pke_set_operand_slot (uint8_t operand,
uint8_t slot_num);

Description:

This routine sets the slot for the selected operand.

Inputs:

Input Parameters	Description
operand	An unsigned 8 bit integer specifying the Operand
slot_num	An unsigned 8 bit integer specifying the slot number

Outputs:

None

4.3.7.2 api_pke_set_operand_slots

Function Header:

void api_pke_set_operand_slots (uint32_t op_slots);

Description:

This routine sets the slots for the operands 0, 1, 2.

Inputs:

Input Parameters	Description
op_slots	An unsigned 32 bit value containing the slot numbers for operands 0, 1, 2 respectively

Outputs:

None

4.3.8 Get Slot Details

4.3.8.1 Get Slot Number

Functions:

api_pke_get_operand_slot

Function Header:

```
uint8_t api_pke_get_operand_slot (uint8_t operand);
```

Description:

This routine returns the slot number for the specified operand.

Inputs:

Input Parameters	Description
operand	An unsigned 8 bit integer specifying the Operand

Outputs:

Slot number

4.3.8.2 Get Slot Address

Function Header:

api_pke_get_slot_addr

Function Header:

```
uint32_t api_pke_get_slot_addr (uint8_t slot_num);
```

Description:

This routine returns the address of the specified slot in the crypto memory

Inputs:

Input Parameters	Description
slot_num	An unsigned 8 bit integer specifying the slot number

Outputs:

Address of the specified slot.

4.3.9 Clear Slots in the Crypto Memory

4.3.9.1 api_pke_scm_clear_slot

Function Header:

```
void api_pke_scm_clear_slot (uint8_t slot_num);
```

Description:

This routine clears a specified slot in SCM by filling it with 0's.

Inputs:

Input Parameters	Description
slot_num	An unsigned 8 bit integer specifying the slot number

Outputs:

None

4.3.9.2 api_pke_clear_scm

Function Header:

```
void api_pke_clear_scm(void);
```

Description:

This function clears the PKE block's shared crypto memory.

Note: The caller must ensure the PKE engine is idle and not in sleep state (clock gated) before calling this routine.

Inputs:

None

Output:

None.

4.3.10 Read from the Crypto Memory

Function:

```
api_pke_copy_from_scm
```

Function Header:

```
uint32_t api_pke_copy_from_scm ( uint8_t* dest,
                                uint32_t nbytes,
                                uint8_t slot_num,
                                bool reverse_byte_order);
```

Description:

This routine is used to read a specified amount of data from the specified SCM slot.

Inputs:

Input Parameter	Description
Dest	An unsigned 8 bit integer pointer to the destination where data will be copied to
Nbytes	An unsigned 32 bit integer specifying the number of bytes to be read.
Slot_num	An unsigned 8 bit integer specifying the slot from which data is to be read
Reverse_byte_order	A Boolean flag which if true reads data in reverse byte order

Outputs:

The routine returns the number of bytes copied to the destination.

4.3.11 Write to the Crypto Memory

4.3.11.1 api_pke_copy_to_scm

Function Header:

```
uint32_t api_pke_copy_to_scm (const uint8_t *pdata,
                             uint32_t nbytes,
                             uint8_t slot_num);
```

Description:

This routine is used to copy bytes from the source to the specified PKE SCM slot with zero fill. The SCM is 24KB with the lower 16KB organized as 32 slots of 512 bytes each.

Inputs:

Input Parameter	Description
Pdata	An unsigned 8 bit integer pointer to the data that is to be written to the slot
nbytes	An unsigned 32 bit integer indicating the number of bytes to be written
Slot_num	An unsigned 8 bit integer indicating the slot number

Outputs:

This function returns the number of bytes copied to the slot.

4.3.11.2 api_pke_fill_slot

Function Header:

```
void api_pke_fill_slot (const uint8_t slot_num,
                       const uint32_t fill_val);
```

Description:

This routine fills the slot memory with fill_val.

Inputs:

Input Parameters	Description
slot_num	An unsigned 8 bit integer specifying the slot number
fill_val	An unsigned 32 bit integer value to be filled in slot memory

Outputs:

None

4.3.11.3 api_pke_copy_to_scm2

Function Header:

```
uint32_t api_pke_copy_to_scm2 (const uint8_t *pdata,
                              uint32_t nbytes,
                              uint8_t slot_num,
```

```
bool reverse_byte_order);
```

Description:

This routine is used to write specified amount of data to a specified SCM slot as DWORD and the option to reverse the byte order. Zero fill is added if the data is not a multiple of 256 bytes.

Inputs:

Input Parameter	Description
Pdata	An unsigned 8 bit integer pointer to data that is to be written into the slot.
Num_bytes	An unsigned 32 bit integer indicating the number of bytes to be written.
Slot_num	An unsigned 8 bit integer indicating the slot number.
Reverse_byte_order	A Boolean flag which if true writes data in reverse byte order.

Outputs:

This function returns the number of bytes that were copied to the specified SCM slot.

4.3.11.4 api_pke_copy_to_scm4

Function Header:

```
uint32_t api_pke_copy_to_scm4 (const uint8_t *pdata,
                               uint32_t nbytes,
                               uint8_t slot_num,
                               uint8_t flags);
```

Description:

This routine is used to write specified amount of data to a specified SCM slot as bytes with optional zero fill and optional byte reversal.

Inputs:

Input Parameter	Description
Pdata	An unsigned 8 bit integer pointer to data to be written
nbytes	An unsigned 32 bit integer indicating the number of words to written
Slot_num	An unsigned 8 bit integer indicating the slot number
flags	An unsigned 8 bit integer having the following bit definitions: <ul style="list-style-type: none"> • b[0]=0 /1 (LSBF/MSBF) • b[1]=0 /1 (no zero fill/zero fill to end of slot)

Outputs:

This function returns the number of bytes copied into the SCM slot.

4.3.12 Read and Clear PKE Status

Functions:

```
api_pke_ists_clear
```

Function Header:

```
uint32_t api_pke_ists_clear(void);
```

Description:

Read and clear Status bit for PKE and the interrupt source bits for PKE block.

Inputs:

None

Outputs:

PKE status[31:0] – in which [31:17] are Reserved; GIRQ16.Source[1:0] in bits[21:20]; PKE status register value in bits [16:0].

4.3.13 Perform Modular Arithmetic

Functions:

api_pke_modular_math

Function Header:

```
uint8_t api_pke_modular_math (uint32_t op_size,
                             const void *P,
                             uint16_t pnbytes,
                             const void *A,
                             uint16_t anbytes,
                             const void *B,
                             uint16_t bnbytes);
```

Description:

This routine is used to perform modular arithmetic.

Inputs:

Input Parameter	Description
op_size	An unsigned 32 bit integer specifying the operand size 64 bits to 4096 bits b[7]= 0/1 (Field F(p)/Field F(2 ^m)) b[15:8]=size in units of 64 bits bit[16]=0/1 (LSBF/ MSBF) b[6:0] =operation Operation in bits[6:0] * 0x00 Reserved * 0x01 C = (A+B) mod P * 0x02 C = (A-B) mod P * 0x03 C = (A*B) mod P (P odd) * 0x04 C = B mod P (P odd), A is ignored * 0x05 C = (A/B) mod P (P odd) * 0x06 C = (1/B) mod P (P odd) * 0x07 Reserved * 0x08 C = (A * B) F(p) only, P is ignored * 0x09 C = (1/B) mod P (P even), A is ignored * 0x0A C = B mod P (P even), A is ignored
P	Pointer to parameter P
pnbytes	An unsigned 16 bit integer specifying the byte length of P
A	Pointer to parameter A
anbytes	An unsigned 16 bit integer specifying the byte length of A
B	Pointer to parameter B
bnbytes	An unsigned 16 bit integer specifying the byte length of parameter B

Outputs:

The return values and their description are presented below.

PKE_RET_ERR_INVALID_BIT_LENGTH – this error value is returned if b[15:8] in op_size has invalid data.

PKE_RET_OK – this value indicates a successful execution of requested operation.

PKE_RET_ERR_BAD_PARAM – this error is returned if 0 bytes were copied into the slot

PKE_RET_ERR_UNKNOWN_OP – this error is returned if op_size have invalid data

PKE_RET_ERR_BUSY – This error value is returned if pke is busy.

Return code macro values:

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BUSY	1
PKE_RET_ERR_BAD_PARAM	2
PKE_RET_ERR_UNKNOWN_OP	4
PKE_RET_ERR_INVALID_BIT_LENGTH	5

4.4 Elliptic Curve Functions

Note: Before using ECDSA functions the PKE block needs to be powered ON explicitly using api_pke_power().

4.4.1 Elliptic Curve Arithmetic Operations

4.4.1.1 EC Point Doubling

Functions:

api_pke_ec_point_double

Function Header:

```
uint8_t api_pke_ec_point_double (const uint8_t* pxy,
                                uint16_t coord_len,
                                bool msbf);
```

Description:

This routine performs the point doubling operation.

Inputs:

Input Parameter	Description
pxy	An unsigned 8 bit Pointer to the point on ec curve
coord_len	An unsigned 16 bit integer specifying the length (in bytes) of the coordinates (px+py)
msbf	A Boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

ECC_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0

ECC_ERR_BUSY	1
ECC_ERR_BAD_PARAM	2

4.4.1.2 EC Point Addition

Functions:

api_pke_ec_point_addition

Function Header:

```
uint8_t api_pke_ec_point_addition (const uint8_t* p1xy,
                                   const uint8_t* p2xy,
                                   uint16_t coord_len,
                                   bool msbf);
```

Description:

This routine performs addition of two points on the EC curve.

Inputs:

Input Parameter	Description
p1xy	An unsigned 8 bit Pointer to the point1 coordinates on the ec curve
p2xy	An unsigned 8 bit Pointer to the point2 coordinates on the ec curve
coord_len	An unsigned 16 bit integer specifying the length (in bytes) of the coordinates (px+py)
msbf	A Boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

ECC_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BUSY	1
ECC_ERR_BAD_PARAM	2

4.4.1.3 EC Point Multiplication

4.4.1.3.1 *api_pke_ec_point_scalar_mult*

Function Header:

```
uint8_t api_pke_ec_point_scalar_mult (const uint8_t* pxy,
                                       uint16_t coord_len,
                                       const uint8_t* pscalar,
                                       uint16_t scalar_len,
                                       uint8_t msbf);
```

Description:

This routine performs scalar point multiplication of an EC curve (P (px,py)) to a scalar data. This API can be used when both the x and y coordinates are present in the same array and the length of the EC curve coordinates is not equal to the length of the scalar data.

Inputs:

Input Parameter	Description
pxy	An unsigned 8 bit Pointer pointing to coordinates px followed by py
coord_len	An unsigned 16 bit integer specifying the length (in bytes) of the EC curve coordinates pxy
pscalar	An unsigned 8 bit Pointer to scalar data to be multiplied with
scalar_len	An unsigned 16 bit integer specifying the length (in bytes) of the scalar data
msbf	An unsigned 8 bit integer which if 1 indicates most significant byte first and if 0 indicates least significant byte first.

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

ECC_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BUSY	1
ECC_ERR_BAD_PARAM	2

4.4.1.3.2 *api_pke_ec_point_scalar_mult2*

Function Header:

```
uint8_t api_pke_ec_point_scalar_mult2 (const uint8_t* px,
                                       const uint8_t* py,
                                       const uint8_t* pscalar,
                                       uint16_t byte_len,
                                       bool msbf);
```

Description:

This routine performs scalar point multiplication of an EC curve (P (px,py)) to a scalar data. This API can be used when the x and y coordinates are given as separate array inputs and the length of the EC curve coordinates is the same as the length of the scalar data.

Inputs:

Input Parameter	Description
px	An unsigned 8 bit Pointer pointing to the px coordinate of the EC curve
py	An unsigned 8 bit Pointer pointing to the py coordinate of the EC curve
pscalar	An unsigned 8 bit Pointer to scalar data to be multiplied with
byte_len	An unsigned 16 bit integer specifying the length (in bytes) of the scalar data
msbf	A Boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

ECC_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BUSY	1
ECC_ERR_BAD_PARAM	2

4.4.1.3.3 *api_pke_ec_point_scalar_mult3*

Function Header:

```
uint8_t api_pke_ec_point_scalar_mult3 (const uint8_t* pxy,
                                       const uint8_t* pscalar,
                                       uint16_t byte_len,
                                       bool msbf);
```

Description:

This routine performs scalar point multiplication of an EC curve (P (px,py)) to a scalar data. This API can be used when both the x and y coordinates are present in the same array and the length of the EC curve coordinates is the same as the length of the scalar data.

Inputs:

Input Parameter	Description
Pxy	An unsigned 8 bit Pointer pointing to coordinates px followed by py
pscalar	An unsigned 8 bit Pointer to scalar data to be multiplied with
byte_len	An unsigned 16 bit integer specifying the length (in bytes) of the scalar data
msbf	A Boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

ECC_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BUSY	1
ECC_ERR_BAD_PARAM	2

4.4.2 Checking Elliptic Curve Parameters

4.4.2.1 Check if point is on the EC Curve

4.4.2.1.1 *api_pke_ec_check_poc2*

Function Header:

Microchip Confidential,

API Manual

```
uint8_t api_pke_ec_check_poc2 (const uint8_t* px,
                               const uint8_t* py,
                               uint16_t plen,
                               bool msbf);
```

Description:

This routine check if the point lies on the EC curve. This API can be used when the x and y coordinates are to be given as separate array inputs.

Inputs:

Input Parameter	Description
Px	An unsigned 8 bit Pointer pointing to px coordinate of the EC curve
py	An unsigned 8 bit Pointer pointing to py coordinate of the EC curve
plen	An unsigned 16 bit integer specifying the length (in bytes) of (px + py)
msbf	A Boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

ECC_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BUSY	1
ECC_ERR_BAD_PARAM	2

4.4.2.1.2 *api_pke_ec_check_poc3*

Function Header:

```
uint8_t api_pke_ec_check_poc3 (const uint8_t* p,
                               uint16_t plen,
                               uint8_t msbf);
```

Description:

This routine check if the point lies on the EC curve. This API can be used when both the x and y coordinates are present in the same array.

Inputs:

Input Parameter	Description
P	An unsigned 8 bit Pointer pointing to coordinates px followed by py
plen	An unsigned 16 bit integer specifying the length (in bytes) of (px + py)
msbf	An unsigned 8 bit value which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

ECC_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BUSY	1
ECC_ERR_BAD_PARAM	2

4.4.2.2 Check Point Coordinates

Functions:

api_pke_ec_check_pxy_less_prime

Function Header:

```
uint8_t api_pke_ec_check_pxy_less_prime (const uint8_t* pxy,
                                         uint16_t plen,
                                         bool msbf);
```

Description:

This routine checks if the Point coordinates are less than the prime.

Inputs:

Input Parameter	Description
Pxy	An unsigned 8 bit Pointer pointing to coordinates (px + py)
plen	An unsigned 16 bit integer specifying the length of the EC curve passed for the coordinates (Px+py)
msbf	A Boolean flag which if true writes data in reverse byte order

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

ECC_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BUSY	1
ECC_ERR_BAD_PARAM	2

4.4.2.3 Check a and b

Functions:

api_pke_ec_check_ab

Function Header:

```
uint8_t api_pke_ec_check_ab (void);
```

Description:

This routine check EC Curve parameters a & b.

Inputs:

None

Outputs:

The return values and their description are presented below.

ECC_ERR_BUSY– This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BUSY	1

4.4.2.4 Check EC Curve Order (n)

Functions:

api_pke_ec_check_n

Function Header:

uint8_t api_pke_ec_check_n (void);

Description:

This routine check EC Curve order (parameter n).

Inputs:

None

Outputs:

The return values and their description are presented below.

ECC_ERR_BUSY– This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BUSY	1

4.4.3 Load Elliptic Curve Parameters into the Crypto Memory

Functions:

api_pke_ec_prog_curve

Function Header:

uint8_t api_pke_ec_prog_curve(const ELLIPTIC_CURVE* curve_p);

Description:

This routine programs the elliptic curve parameters into the PKE shared crypto memory. All parameters are zero extended to end of the slot. The PKE slots size is 512 bytes. This routine also programs the following register fields in command register.

Input Parameters	Description
Curve_p	<p>A pointer to ELLIPTIC_CURVE structure. The structure definition is presented below.</p> <pre> Struct ELLIPTIC_CURVE{ Uint32_t * param[ELLIPTIC_CURVE_NPARAMS]; Uint16_t byte_len; Uint8_t flags; Uint8_t rsvd1; }; </pre> <p>Curve parameters are most-significant-byte first. EC firmware will byte reverse before writing to PKE SCM.</p> <pre> #define EC_FLAG_LSB (0u << 0) #define EC_FLAG_MSB (1u << 0) #define EC_FLAG_F2M (1u << 1) // Curve is binary </pre> <p>Elliptic Curve parameters a and b can be supplied in a negative encoding OR in a positive encoding. If the parameter is negative but encoded as positive then we use these flags to inform the PKE engine to negate the parameter before use. For example, common curves use a = -3. It's usually supplied in negative encoding with leading 1's. We could encode it as 3 (0x0000....0003) and use C_FLAG_ANEG to configure PKE to negate the value.</p> <pre> #define EC_FLAG_ANEG (1u << 2) #define EC_FLAG_BNEG (1u << 3) </pre> <p>PKE stored curve parameters in units of 64-bits(8-bytes) padded with zeros.</p> <pre> #define EC_192_PKE_LEN (192ul / 8ul) #define EC_224_PKE_LEN (224ul / 8ul) #define EC_256_PKE_LEN (256ul / 8ul) #define EC_384_PKE_LEN (384ul / 8ul) #define EC_512_PKE_LEN (512ul / 8ul) </pre> <p>NOTE: PKE HW must round P-521 length up to next even multiple of 64-bits (10)</p> <pre> #define EC_640_PKE_LEN (640ul / 8ul) </pre> <p>Actual Elliptic curve parameter lengths. The above ELLIPTIC_CURVE_xxx_LEN parameters are the size of the PKE engine slots used for the curve. Slot lengths are in units of 8-bytes and may be larger than actual curve parameters. PKE requires zero padding of data smaller than the slot length. The symbols below are the actual curve coordinate byte lengths.</p> <pre> #define EC_P192_LEN (0x18ul) // same as above, multiple of 64 #define EC_P224_LEN (0x1Cul) // not a multiple of 64 #define EC_P256_LEN (0x20ul) // same as above, multiple of 64 #define EC_P384_LEN (0x30ul) // same as above, multiple of 64 #define EC_P521_LEN (0x42ul) // not a multiple of 64 // #define EC_B163_LEN (0x15ul) #define EC_B233_LEN (0x1Eul) #define EC_B283_LEN (0x24ul) #define EC_B409_LEN (0x34ul) #define EC_B571_LEN (0x48ul) </pre> <p>The order of parameters and their description is presented below.</p> <p>ELLIPTIC_CURVE_NPARAMS – 6u (max parameters - 7)</p> <p>0 – ELLIPTIC_CURVE_PARAM_P</p> <p>1 – ELLIPTIC_CURVE_PARAM_N</p>

	2 – ELLIPTIC_CURVE_PARAM_GX 3 – ELLIPTIC_CURVE_PARAM_GY 4 – ELLIPTIC_CURVE_PARAM_A 5 – ELLIPTIC_CURVE_PARAM_B The indices of parameters correspond to SCM slot numbers.
--	---

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – This error value is returned if curve_p points to NULL or if any of the pointers in the curve parameters point to NULL.

PKE_RET_OK – if the requested operation is successful.

Return code macro values:

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BAD_PARAM	2

4.4.4 ECDSA Signature Verification

Functions:

api_pke_ecdsa_verify3

Function Header:

```
uint8_t api_pke_ecdsa_verify3 (const uint8_t* Q,
                               const uint8_t* S,
                               const uint8_t* digest,
                               uint16_t elen,
                               uint16_t dlen,
                               bool msbf);
```

Description:

This routine verifies a signature using standard EC Digital Signature Algorithm.

Inputs:

Input Parameters	Description
Q	An unsigned 8 bit integer pointer to constant data containing Public Key Q
S	An unsigned 8 bit integer pointer to constant data consisting of signature point S
digest	An unsigned 8 bit integer pointer to constant data consisting of hash digest of the message
Elen	An unsigned 16 bit integer specifying the length of Qx and Qy. (in bytes)
dlen	An unsigned 16 bit integer specifying the length of digest (in bytes)
Msbf	A Boolean value is true, indicates that all parameters are most significant byte first.

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

ECC_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Return code macro values:

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BUSY	1
ECC_ERR_BAD_PARAM	2

4.4.5 KCDSA Functions

4.4.5.1 Generate KCDSA Key

Functions:

api_pke_ec_kcdda_keygen

Function header:

```
uint8_t api_pke_ec_kcdda_keygen (const uint8_t* d,
                                uint16_t plen,
                                uint16_t flags);
```

Description:

This routine generates EC public key. Caller must have previously programmed elliptic curve into the SCM using api_pke_ec_prog_curve(). The output can be read from:

Output parameter	Slot Number
Qx (public key x-coordinate)	Slot 8
Qy (public key y-coordinate)	Slot 9

Inputs:

Input Parameters	Description
d	An unsigned 8 bit pointer to the array containing the EC private key
plen	An unsigned 16 bit integer specifying the length of d (in bytes)
flags	An unsigned 16 bit integer specifying the Byte order of d. bit[0]=0/1 (LSBF/ MSBF)

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid or if point parameter length is greater than curve parameter length.

ECC_ERR_BUSY – This error value is returned if pke is busy.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BUSY	1

ECC_ERR_BAD_PARAM	2
-------------------	---

4.4.5.2 KCDSA Signature Generation

Functions:

api_pke_ec_kcdda_sign

Function header:

```
uint8_t api_pke_ec_kcdda_sign (const uint8_t* prv_key,
                               uint16_t plen,
                               const uint8_t* r,
                               uint16_t rlen,
                               const uint8_t* hash,
                               uint16_t hlen,
                               uint16_t flags);
```

Description:

This routine performs the KCDSA signature generation operation. Caller must have previously programmed elliptic curve into the SCM using api_pke_ec_prog_curve().

Inputs:

Input Parameters	Description
Prv_key	An unsigned 8 bit pointer to the array containing the EC private key
plen	An unsigned 16 bit integer specifying the length of prv_key (in bytes)
r	An unsigned 8 bit Pointer to array containing the r component of the signature
rlen	An unsigned 16 bit integer specifying the length of r (in bytes)
hash	An unsigned 8 bit pointer to hash digest of message
hlen	An unsigned 16 bit integer specifying the length of hash digest (in bytes)
flags	An unsigned 16 bit integer specifying the following: bit[0]=0(prv_key is LSBF), 1(prv_key is MSBF) bit[1]=0(r is LSBF), 1(r is MSBF) bit[2]=0(hash digest is LSBF), 1(hash digest is MSBF)

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid or if point parameter length is greater than curve parameter length.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BAD_PARAM	2

4.4.5.3 KCDSA Signature Verification

Functions:

api_pke_ec_kcdda_verify

Function header:

```
uint8_t api_pke_ec_kcdsa_verify (const uint8_t* q,
                                uint16_t qlen,
                                const uint8_t* sig,
                                uint16_t slen,
                                const uint8_t* hash,
                                uint16_t hlen,
                                uint16_t flags);
```

Description:

This routine performs signature verification operation. Caller must have previously programmed elliptic curve into the SCM using `api_pke_ec_prog_curve()`.

Inputs:

Input Parameters	Description
Q	An unsigned 8 bit pointer to the array containing the EC public key Qx and Qy
qlen	An unsigned 16 bit integer specifying the length of Q (in bytes)
Sig	An unsigned 8 bit pointer to the array containing r & s
slen	An unsigned 16 bit integer specifying the length of the signature (in bytes)
hash	An unsigned 8 bit pointer to the array containing the hash digest of the message
hlen	An unsigned 16 bit integer specifying the length of the hash digest (in bytes)
Msbf	An unsigned 16 bit integer specifying the following: bit[0]=0(Qx,y are LSBF), 1(Qx,y are MSBF) bit[1]=0(r,s are LSBF), 1(r,s are MSBF) bit[2]=0(digest is LSBF), 1(digest is MSBF)

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid or if point parameter length is greater than curve parameter length.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BAD_PARAM	2

4.4.6 SRP Operation

Functions:

`api_pke_srp_sc`

Function Header:

```
uint8_t api_pke_srp_sc (const PKE_SRP_DATA *psrp);
```

Description:

This routine uses PKE to generate step 4 of SRP algorithm. The length provided in the inputs is in units of 64 bytes and in the range [0x02, 0x40]. The output can be read from Slot B.

Inputs:

Input Parameters	Description
Psrp	<p>A pointer to the PKE_SRP_DATA structure containing the length and pointers to each of the seven parameters. The structure definition is presented below.</p> <pre>struct PKE_SRP_DATA{ uint16_t len64b; uint16_t flags; uint8_t * param[PKE_MAX_SRP_PARAM]; };</pre> <p>Where, len64b – length of P in blocks of 64 bits. The permitted values of flags are PKE_SRP_FLAG_LSBF – 0 PKE_SRP_FLAG_MSBF – 1</p> <p>The bit definitions of param are listed below. PKE_MAX_SRP_PARAM – The maximum number of parameters (7) PKE_SRP_PARAM_P – 0 PKE_SRP_PARAM_G – 1 PKE_SRP_PARAM_A – 2 PKE_SRP_PARAM_B – 3 PKE_SRP_PARAM_X – 4 PKE_SRP_PARAM_K – 5 PKE_SRP_PARAM_U – 6</p> <p>All the input parameters (P,G,a,B,X,K,U) should be of length len64b. If they are not, pad them with zeros wherever necessary.</p>

Outputs:

The return values and their descriptions are presented below.

PKE_RET_ERR_BAD_ADDR – this error value is returned if input parameters points to NULL or if length parameter is invalid (see description).

PKE_RET_OK – this value is returned if requested operation was successful

Macro Name	Value
PKE_RET_OK	0
PKE_RET_ERR_BAD_ADDR	3

4.4.7 Elliptic Curve 25519 Point Multiplication

Functions:

api_pke_ec25519_ptmult

Function header:

```
uint8_t api_pke_ec25519_ptmult (const uint8_t* p1x,
                                uint16_t p1x_len,
                                const uint8_t* k,
                                uint16_t k_len,
                                uint16_t flags);
```

Description:

This routine performs Elliptic Curve25519 Point Multiplication

Inputs:

Input Parameters	Description
P1x	An unsigned 8 bit pointer to the array containing the parameter px
P1x_len	An unsigned 16 bit integer specifying the length of p1x (in bytes)
K	An unsigned 8 bit pointer to the array containing the parameter K
K_len	An unsigned 16 bit integer specifying the length of k (in bytes)
flags	An unsigned 16 bit integer specifying the following: bit[0] = 0(p1x byte array is LSBF), 1(MSBF) bit[1] = 0(k byte array is LSBF), 1(MSBF)

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BAD_PARAM	2

4.4.8 Edward's Curve 25519 Operations

4.4.8.1 ED25519 Scalar Multiplication

Functions:

api_pke_ed25519_scalarmult

Function header:

```
uint8_t api_pke_ed25519_scalarmult (const uint8_t* px,
                                     uint16_t pxlen,
                                     const uint8_t* py,
                                     uint16_t pylen,
                                     const uint8_t* e,
                                     uint16_t elen,
                                     uint16_t flags);
```

Description:

This function multiplies a point by the specified scalar and stores the result at Slot[0xA]=x-coordinate and Slot[0xB]=y-coordinate in the SCM.

Inputs:

Input Parameters	Description
Px	An unsigned 8 bit Pointer pointing to the parameter px
Pxlen	An unsigned 16 bit integer specifying the length of px (in bytes)
Py	An unsigned 8 bit Pointer pointing to the parameter py
Pylen	An unsigned 16 bit integer specifying the length of py (in bytes)
E	An unsigned 8 bit Pointer pointing to the parameter e
Elen	An unsigned 16 bit integer specifying the length of e (in bytes)

flags	An unsigned 16 bit integer specifying the following: bit[0] = 0(px & py are LSBF), 1(MSBF) bit[1] = 0(e is LSBF), 1(MSBF)
-------	---

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BAD_PARAM	2

4.4.8.2 ED25519 Recover X-Coordinate

Function header:

```
uint8_t api_pke_ed25519_xrecov (const uint8_t* y,
                                uint16_t ylen,
                                uint16_t flags);
```

Description:

This routine recovers the X-coordinate given Y-coordinate

Inputs:

Input Parameters	Description
y	An unsigned 8 bit pointer to the array containing the parameter y
ylen	An unsigned 16 bit integer specifying the length of y (in bytes)
flags	An unsigned 16 bit integer specifying the following: 0 (y is LSBF), 1(y is MSBF)

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BAD_PARAM	2

4.4.8.3 ED25519 Signature Verification

Functions:

```
api_pke_ed25519_valid_sig
```

Function header:

```
uint8_t api_pke_ed25519_valid_sig (const Ed25519_SIG_VERIFY* psv);
```


Description:

This function checks ED25519 signature (point) against message string (hash)

Inputs:

Input Parameters	Description
psv	<p>pointer to structure Ed25519_SIG_VERIFY</p> <pre>#define ED_PARAM_AX (0u) #define ED_PARAM_AY (1u) #define ED_PARAM_RX (2u) #define ED_PARAM_RY (3u) #define ED_PARAM_SIG (4u) #define ED_PARAM_HASH (5u) #define ED_PARAM_MAX (6u)</pre> <pre>typedef struct { uint8_t* params[ED_PARAM_MAX]; uint16_t paramlen[ED_PARAM_MAX]; uint16_t flags; uint16_t rsvd; } Ed25519_SIG_VERIFY;</pre> <p>Load appropriate parameter to the params[] array Params[ED_PARAM_AX] = pointer to the Param Ax Params[ED_PARAM_AY] = pointer to the Param Ay Params[ED_PARAM_RX] = pointer to the Param Rx Params[ED_PARAM_RY] = pointer to the Param Ry Params[ED_PARAM_SIG] = pointer to the Param Signature Params[ED_PARAM_HASH] = pointer to the Param A</p> <p>Flags - A Boolean flag which if true writes data in reverse byte order Bit[0] corresponds to Ax Bit[1] corresponds to Ay Bit[2] corresponds to Rx Bit[3] corresponds to Ry Bit[4] corresponds to Signature Bit[5] corresponds to Hash</p>

Outputs:

The return values and their description are presented below.

ECC_ERR_BAD_PARAM – this error value is returned if input parameters are invalid.

PKE_RET_OK – this value indicates a successful execution of requested operation

Macro Name	Value
ECC_OK / PKE_RET_OK	0
ECC_ERR_BAD_PARAM	2

4.5 RNG Functions

4.5.1 Powering the Block On/Off

Functions:

api_rng_power

Function Header:

```
Void api_rng_power(bool pwr_on);
```

Description:

This routine is used for powering on/off the RNG block.

Inputs:

Input Parameters	Description
Pwr_on	A Boolean value if false puts the module to sleep (gate off clocks to block), if true enables the block (gate on clocks to block)

Outputs:

None

4.5.2 Reset the Block

Functions:

api_rng_reset

Function Header:

```
Void api_rng_reset(void);
```

Description:

This routine resets the hardware of the RNG block to the hardware default.

Inputs:

None

Outputs:

None

4.5.3 Set the Mode of Operation

Functions:

api_rng_mode

Function Header:

```
Void api_rng_mode(uint8_t tmode_pseudo);
```

Description:

The function controls the mode of RNG. The possible modes are Asynchronous (true random mode), and pseudo random mode.

Inputs:

Input Parameters	Description
Tmode_pseudo	An 8 bit unsigned integer if zero, enables asynchronous mode and if 1 enables

pseudo random mode

Outputs:

None

4.5.4 Check if the Block has Started

Functions:

api_rng_is_on

Function Header:

Bool api_rng_is_on(void);

Description:

This function is used to check if the NDRNG block has started.

Inputs:

None

Outputs:

Returns true if block is on, false otherwise.

4.5.5 Start the block

Functions:

api_rng_start

Function Header:

Void api_rng_start(void);

Description:

This routine is used to start the NDRNG engine. Once started, the NDRNG will fill its internal 1Kbit internal FIFO with random bits. The NDRNG block will hang if its FIFO is read while empty. Firmware must poll the NDRNG's FIFO level and only read data from the 32-bit FIFO data register when NOT empty.

Inputs:

None

Outputs:

None

4.5.6 Stop the block

Functions:

api_rng_stop

Function Header:

Void api_rng_stop(void);

Description:

This routine stops the NDRNG engine. When the engine is stopped, the NDRNG will not re-fill its FIFO when data is removed.

Inputs:

None

Outputs:

None

4.5.7 Get Data from FIFO

Functions:

api_rng_get_num_random_words

Function Header:

```
Uin32_t api_rng_get_num_random_words(void);
```

Description:

This routine reads the NDRNG FIFO level register and returns the number of 32-bit words of random data currently in FIFO. This call must be issued before reading the FIFO and only read FIFO if this call returns a non-zero number.

Inputs:

None

Outputs:

The call returns the number of 32-bit words in the NDRNG FIFO. Maximum value is 32 (32x32 = 1024 bits).

4.5.8 Retrieve Data as Bytes from FIFO

Functions:

api_rng_get_random_bytes

Function Header:

```
uint32_t api_rng_get_random_bytes ( uint8_t *pbuff8,
                                     uint32_t num_bytes);
```

Description:

This routine fills a buffer with random bytes. The API reads 32 bits at a time from FIFO. If bytes are requested, a 32 bit word is read and 4 bytes are retrieved. However, only the number of bytes requested is returned.

Inputs:

Input Parameters	Description
Pbuff8	An unsigned 8 bit integer pointer to a buffer where the data will be stored
Num_bytes	An unsigned 32 bit integer indicating the number of random bytes to be retrieved. For api_rng_get_random_bytes, it must be less than or equal to the size of the buffer.

Output:

The number of bytes retrieved is returned.

4.5.9 Retrieve Data as Words from FIFO

Functions:

api_rng_get_random_words

Function Header:

```

uint32_t api_rng_get_random_words (uint32_t *pbuff32,
                                   uint32_t num_words);

```

Description:

This function reads a specified number of words (32-bit data) into the buffer specified by the caller. If the Random number generator is stopped or no data is available in the FIFO, it will return zero.

Note: If the NDRNG FIFO hardware stops filling, then this routine will loop forever.

Inputs:

Input Parameters	Description
Pbuff32	An unsigned 32 bit integer pointer to a 32-bit aligned SRAM buffer
Num_words	An unsigned 32 bit integer indicating the number of 32-bit words of random data to read

Output:

Returns the actual number of words read.

4.6 HASH Functions

Note 1: Before using hash functions, the hash block needs to be powered ON using `api_aes_hash_power()`.

4.6.1 Check the Status of the Block

Functions:

`api_hash_status`

Function Header:

```

uint32_t api_hash_status (void);

```

Description:

This routine returns the status of the hash block.

Inputs:

None

Outputs:

Returns the status of the HASH block. The status register is read only. The bit definition is provided below.

Bit Number	Description
0	This bit reflects AHB error. If 0, there is no error. If 1, it indicates that AHB error has occurred.

4.6.2 Check if the Block is Busy

Functions:

`api_hash_busy`

Function Header:

```
Bool api_hash_busy (void);
```

Description:

This routine is used to check if the HASH block is busy.

Inputs:

None

Outputs:

Returns a Boolean value which, if true, indicates that the block is busy.

4.6.3 Starting the Hash Block

Functions:

```
api_hash_start
```

Function Header:

```
void api_hash_start (uint8_t ien);
```

Description:

This routine is used to start the HASH engine. Once started, the GIRQ16 bit 4 reflects the done status (1 if done). It must be cleared after it is set.

Inputs:

Input Parameters	Description
ien	An unsigned 8 bit value indicating the state of interrupts. This value, if true enables interrupt and if false, disables interrupt.

Outputs:

None

4.6.4 Check the Done Status

4.6.4.1 api_hash_is_done_status

Function Header:

```
uint8_t api_hash_is_done_status (uint32_t *status_value);
```

Description:

This routine is used to check the done status of HASH block. The status register value is updated into the pointer passed by the buffer.

Input:

Input Parameter	Description		
Status_value	An unsigned 32 bit integer pointer where the status value will be stored. The bit definitions are listed below		
	<table> <tr> <th>Bit</th><th>Description</th></tr> </table>	Bit	Description
Bit	Description		

	Numbers	
	Bit[16]	This bit indicates if the core is busy or not. If the value is 1, the core is busy; 0 otherwise.
	Bit[1]	This bit indicates if the block has started or not. If the value is 1, operation has started; 0 otherwise.

Outputs:

The return value is 1 if the operation is done, 0 otherwise.

4.6.4.2 api_hash_is_done_status2

Function Header:

```
uint8_t api_hash_is_done_status2 (uint32_t *status_value);
```

Description:

This function returns true if the hash engine has completed the operation and has stopped. The value of the status bits are stored in the input parameter passed.

Input:

Input Parameter	Description								
status_value	<p>An unsigned 32 bit integer pointer that points to the memory location where the values of the status bits will be stored. The bit definitions of this memory location are as follows:</p> <table> <tr> <th>Bit Numbers</th><th>Description</th></tr> <tr> <td>Bit[15]</td><td>1 – Hash engine finished its current operation 0 – Hash engine busy</td></tr> <tr> <td>Bit[4]</td><td>This bit stores the value of the Hash Status Bit. 1 – AHB master interface error 0 – No error</td></tr> <tr> <td>Bit[0]</td><td>This bit stores the value of the Hash Start Bit. 1 – Hash Engine is busy 0 – Hash engine is idle</td></tr> </table>	Bit Numbers	Description	Bit[15]	1 – Hash engine finished its current operation 0 – Hash engine busy	Bit[4]	This bit stores the value of the Hash Status Bit. 1 – AHB master interface error 0 – No error	Bit[0]	This bit stores the value of the Hash Start Bit. 1 – Hash Engine is busy 0 – Hash engine is idle
Bit Numbers	Description								
Bit[15]	1 – Hash engine finished its current operation 0 – Hash engine busy								
Bit[4]	This bit stores the value of the Hash Status Bit. 1 – AHB master interface error 0 – No error								
Bit[0]	This bit stores the value of the Hash Start Bit. 1 – Hash Engine is busy 0 – Hash engine is idle								

Outputs:

The return value is 1 if the operation is done, 0 otherwise.

4.6.5 Clear Hash Interrupts

Functions:

```
api_hash_iclr
```

Function Header:

```
uint32_t api_hash_iclr(void);
```

Description:

This function is used to clear hash interrupts.

Inputs:

None

Output:

The return value is the status register of hash block.

Hash status – bit 0 = 1 (hash block is busy, status bit set), bit 0 = 0 (hash status is clear).

4.7 SHA Functions

Note 1: Before using SHA functions, the hash block needs to be powered ON using `api_aes_hash_power()`.

4.7.1 SHA Direct Functions

This API can be used only for input messages greater than 64 bytes.

4.7.1.1 Initialize

Function Header:

```
uint8_t api_sha_direct_init (uint8_t sha_mode,
                             uint32_t *pbuff);
```

Description:

This function initializes the buffer for the specified SHA operation.

Inputs:

Input Parameters	Description
sha_mode	An unsigned 8 bit integer specifying the SHA mode. The following macros can also be used: <ul style="list-style-type: none"> • SHA_MODE_1 • SHA_MODE_256 • SHA_MODE_512
pbuff	An unsigned 32 bit integer pointer to the buffer containing the SHA digest and padding. SHA-1/256 = 96 bytes. bytes [0:19]=SHA-1 digest, bytes[0:31]=SHA-2 digest, bytes[32:95]=padding. SHA-512 = 192 bytes. bytes[0:63]=digest, bytes[64:191]=padding

Outputs:

The return values and their description are presented below.

SHA_RET_ERR_UNSUPPORTED – this error is returned if sha_mode parameter has an invalid value

SHA_RET_ERR_BUSY – this value is returned if the hash block is busy.

SHA_RET_ERR_NULL_PTR – this error is returned if pbuff points to NULL.

SHA_RET_ERR_MISALIGNED_DATA – this error is returned if pbuff is not aligned

SHA_RET_OK – this value indicates a successful execution of requested operation.

Macro Name	Value
SHA_RET_OK	0
SHA_RET_ERR_BUSY	0x80
SHA_RET_ERR_UNSUPPORTED	0x84
SHA_RET_ERR_MISALIGNED_DATA	0x85
SHA_RET_ERR_NULL_PTR	0x86

4.7.1.2 Update

Function Header:

```
uint8_t api_sha_direct_update (uint8_t sha_mode,
                               uint32_t *pbuff, uint32_t *pblocks,
                               uint32_t nblocks, uint32_t flags);
```

Description:

This routine programs hash engine with data address and the number of data blocks to process. The SHA block must be initialized before calling this function.

Inputs:

Input Parameters	Description
sha_mode	An unsigned 8 bit integer specifying the SHA mode. The following macros can also be used: <ul style="list-style-type: none"> SHA_MODE_1 SHA_MODE_256 SHA_MODE_512
pbuff	An unsigned 32 bit integer pointer to a buffer containing the SHA digest and padding
pblocks	An unsigned 32 bit pointer to a word (32-bit) aligned buffer containing the input data
nblocks	An unsigned 32 bit integer specifying the number of 64 byte blocks in pblocks
flags	An unsigned 32 bit integer specifying the following: <ul style="list-style-type: none"> b[0] = 0/1 (do not start/start) b[1] = 0/1 (if start, do not enable interrupts/ if start, enable interrupts)

Outputs:

The return values and their description are presented below.

0(OK not started), 1(OK, started), 0x8x(error)

SHA_RET_ERR_UNSUPPORTED – this error is returned if sha_mode parameter has an invalid value

SHA_RET_ERR_BUSY – this value is returned if the hash block is busy.

SHA_RET_ERR_NULL_PTR – this error is returned if the buffers point to NULL.

SHA_RET_ERR_MISALIGNED_DATA – this error is returned if the buffers are not aligned

SHA_RET_OK – this value indicates a successful execution of requested operation.

Macro Name	Value
SHA_RET_OK	0
SHA_RET_START	1
SHA_RET_ERR_BUSY	0x80
SHA_RET_ERR_UNSUPPORTED	0x84
SHA_RET_ERR_MISALIGNED_DATA	0x85
SHA_RET_ERR_NULL_PTR	0x86

4.7.1.3 Implement Padding

Function Header:

```
uint8_t api_sha_direct_finalize (uint8_t sha_mode,
                                  uint32_t *pbuff,
                                  uint32_t total_mesg_byte_len,
                                  uint32_t nrem,
                                  uint8_t *prebytes);
```

Description:

This routine implements the standard SHA padding described in the FIPS standard. On entry, pbuff[0:N] contains intermediate digest value where N is 19(SHA-1), 31(SHA-256), or 63(SHA-512). On exit, pbuff[0:N] contains the final digest value. Remaining bytes of pbuff are used for FIPS padding.

Inputs:

Input Parameters	Description
sha_mode	An unsigned 8 bit integer indicating the mode. Permitted modes are SHA_MODE_1, SHA_MODE_256 and SHA_MODE_512.
pbuff	An unsigned 32 bit integer pointer to a buffer containing the SHA digest and padding
total_mesg_byte_len	An unsigned 32 bit integer specifying the length of the message in bytes.
nrem	An unsigned 32 bit integer specifying the number of remaining message bytes. This value must be less than 64.
prebytes	An unsigned 8 bit integer pointer to the remaining bytes

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_UNSUPPORTED - this error is returned if sha_mode parameter has an invalid value

SHA_RET_ERR_BUSY - this value is returned if the hash block is busy.

SHA_RET_ERR_NULL_PTR - this error is returned if the buffers point to NULL.

SHA_RET_ERR_MISALIGNED_DATA - this error is returned when prebytes is not aligned

SHA_RET_ERR_MAX_REM - this value is returned when the value at nrem is equal to or exceeds the maximum value.

SHA_RET_OK - this value indicates a successful execution of requested operation.

Macro Name	Value
SHA_RET_OK	0
SHA_RET_ERR_BUSY	0x80
SHA_RET_ERR_UNSUPPORTED	0x84
SHA_RET_ERR_MISALIGNED_DATA	0x85
SHA_RET_ERR_NULL_PTR	0x86
SHA_RET_ERR_MAX_REM	0x87

4.7.2 SHA1 Functions

4.7.2.1 Initialize

Function Header:

```
uint8_t api_sha1_init (SHA12_CONTEXT_T* sha12_ctx );
```

Description:

This function initializes the SHA12_CONTEXT_T data structure for SHA1.

Inputs:

Input Parameters	Description
sha12_ctx	A pointer to the SHA12_CONTEXT_T data structure. The structure definition is given below: typedef struct sha12_context_s SHA12_CONTEXT_T;

	<pre> struct sha12_context_s { union { uint32_t w[SHA256_WLEN]; uint8_t b[SHA256_BLEN]; } digest; uint32_t mode; uint32_t block_len; uint64_t msg_byte_len; union { uint32_t w[SHA256_BLOCK_WLEN]; uint8_t b[SHA256_BLOCK_BLEN]; } block; }; </pre>
--	---

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_BAD_ADDR – This error is returned if the input points to NULL.

SHA_RET_OK – This value indicates a successful execution of requested operation.

Macro Name	Value
SHA_RET_OK	0
SHA_RET_ERR_BAD_ADDR	0x81

4.7.2.2 Update

Function Header:

```

uint8_t api_sha1_update (SHA12_CONTEXT_T * sha12_ctx,
    const uint8_t *data,
    uint32_t nbytes);

```

Description:

This routine programs hash engine with data address and the number of data blocks to process.

Inputs:

Input Parameters	Description
sha12_ctx	<p>A pointer to the SHA12_CONTEXT_T data structure. The structure definition is given below:</p> <pre> typedef struct sha12_context_s SHA12_CONTEXT_T; struct sha12_context_s { union { uint32_t w[SHA256_WLEN]; uint8_t b[SHA256_BLEN]; } digest; uint32_t mode; uint32_t block_len; uint64_t msg_byte_len; union { uint32_t w[SHA256_BLOCK_WLEN]; uint8_t b[SHA256_BLOCK_BLEN]; } block; }; </pre>
data	An unsigned 8 bit pointer to the data.

nbytes	An unsigned 32 bit integer specifying the length of data (in bytes)
--------	---

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_NULL_PTR– This error is returned if the input pointers point to NULL.

SHA_RET_OK – This value indicates a successful execution of requested operation.

SHA_RET_ERR_BUSY – this value is returned if the hash block is busy.

Macro Name	Value
SHA_RET_OK	0
SHA_RET_ERR_BUSY	0x80
SHA_RET_ERR_NULL_PTR	0x86

4.7.2.3 Implement Padding

Function Header:

```
uint8_t api_sha1_finalize (SHA12_CONTEXT_T * sha12_ctx);
```

Description:

This routine implement the standard SHA padding described in the FIPS standard.

Inputs:

Input Parameters	Description
sha12_ctx	<p>A pointer to the SHA12_CONTEXT_T data structure. The structure definition is given below:</p> <pre>typedef struct sha12_context_s SHA12_CONTEXT_T; struct sha12_context_s { union { uint32_t w[SHA256_WLEN]; uint8_t b[SHA256_BLEN]; } digest; uint32_t mode; uint32_t block_len; uint64_t msg_byte_len; union { uint32_t w[SHA256_BLOCK_WLEN]; uint8_t b[SHA256_BLOCK_BLEN]; } block; };</pre>

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_NULL_PTR – this value is returned when sha12_ctx points to NULL.

SHA_RET_ERR_BUSY – this value is returned if the hash block is busy.

SHA_RET_OK - This value indicates a successful execution of requested operation.

Macro Name	Value
SHA_RET_OK	0

SHA_RET_ERR_BUSY	0x80
SHA_RET_ERR_NULL_PTR	0x86

4.7.3 SHA256 Functions

4.7.3.1 Initialize

Function Header:

```
uint8_t api_sha256_init (SHA12_CONTEXT_T* sha12_ctx);
```

Description:

This function initializes the SHA12_CONTEXT_T data structure for SHA1.

Inputs:

Input Parameters	Description
sha12_ctx	<p>A pointer to the SHA12_CONTEXT_T data structure. The structure definition is given below:</p> <pre>typedef struct sha12_context_s SHA12_CONTEXT_T; struct sha12_context_s { union { uint32_t w[SHA256_WLEN]; uint8_t b[SHA256_BLEN]; } digest; uint32_t mode; uint32_t block_len; uint64_t msg_byte_len; union { uint32_t w[SHA256_BLOCK_WLEN]; uint8_t b[SHA256_BLOCK_BLEN]; } block; };</pre>

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_BAD_ADDR – This error is returned if the input points to NULL.

SHA_RET_OK – This value indicates a successful execution of requested operation.

Macro Name	Value
SHA_RET_OK	0
SHA_RET_ERR_BAD_ADDR	0x81

4.7.3.2 Update

Function Header:

```
uint8_t api_sha256_update (SHA12_CONTEXT_T * sha12_ctx,
                           const uint8_t *data,
                           uint32_t nbytes);
```

Description:

Microchip Confidential,

API Manual

This routine programs hash engine with data address and the number of data blocks to process.

Inputs:

Input Parameters	Description
sha12_ctx	A pointer to the SHA12_CONTEXT_T data structure. The structure definition is given below: <pre>typedef struct sha12_context_s SHA12_CONTEXT_T; struct sha12_context_s { union { uint32_t w[SHA256_WLEN]; uint8_t b[SHA256_BLEN]; } digest; uint32_t mode; uint32_t block_len; uint64_t msg_byte_len; union { uint32_t w[SHA256_BLOCK_WLEN]; uint8_t b[SHA256_BLOCK_BLEN]; } block; };</pre>
data	An unsigned 8 bit pointer to the data.
nbytes	An unsigned 32 bit integer specifying the length of data (in bytes)

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_NULL_PTR– This error is returned if the input pointers point to NULL.

SHA_RET_OK – This value indicates a successful execution of requested operation.

SHA_RET_ERR_BUSY – this value is returned if the hash block is busy.

Macro Name	Value
SHA_RET_OK	0
SHA_RET_ERR_BUSY	0x80
SHA_RET_ERR_NULL_PTR	0x86

4.7.3.3 Implement Padding

Function Header:

```
uint8_t api_sha256_finalize (SHA12_CONTEXT_T * sha12_ctx);
```

Description:

This routine implement the standard SHA padding described in the FIPS standard.

Inputs:

Input Parameters	Description
sha12_ctx	A pointer to the SHA12_CONTEXT_T data structure. The structure definition is given below: <pre>typedef struct sha12_context_s SHA12_CONTEXT_T; struct sha12_context_s {</pre>

	<pre> union { uint32_t w[SHA256_WLEN]; uint8_t b[SHA256_BLEN]; } digest; uint32_t mode; uint32_t block_len; uint64_t msg_byte_len; union { uint32_t w[SHA256_BLOCK_WLEN]; uint8_t b[SHA256_BLOCK_BLEN]; } block; }; </pre>
--	--

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_NULL_PTR – this value is returned when sha12_ctx points to NULL.

SHA_RET_ERR_BUSY – this value is returned if the hash block is busy.

SHA_RET_OK - This value indicates a successful execution of requested operation.

Macro Name	Value
SHA_RET_OK	0
SHA_RET_ERR_BUSY	0x80
SHA_RET_ERR_NULL_PTR	0x86

4.7.3.4 SHA256 for Small Messages

Function Header:

```

uint8_t api_sha256_under56 (uint32_t *pdigest,
                           uint32_t *pdata56,
                           uint8_t nbytes,
                           uint8_t flags);

```

Description:

This function computes SHA256 for input data less than 56 bytes.

Inputs:

Input Parameters	Description
pdigest	An unsigned 32 bit integer pointer to digest.
pdata56	An unsigned 32 bit integer pointer to the input data which is less than 56 bytes.
nbytes	An unsigned 8 bit integer specifying the length of pdata56 in bytes.
Flags	An unsigned 8 bit integer specifying the following: <ul style="list-style-type: none"> b[0] = 0/1 (do not start/start) b[1] = 0/1 (if start, do not enable interrupts/ if start, enable interrupts)

Outputs:

The return values and their descriptions are presented below.

0 - Failure

1 – Success

4.7.3.5 SHA256 Block Functions

The following functions can be used for SHA 256 if the input data is in multiples of 64 bytes

4.7.3.5.1 *Initialize*

Function Header:

```
uint8_t api_sha256_block_init (uint32_t *pdigest);
```

Description:

This routine initializes the hash engine for SHA256 operation.

Inputs:

Input Parameters	Description
pdigest	An unsigned 32 bit integer pointer to digest

Outputs:

The return values and their descriptions are presented below.

0 - Failure

1 - Success

4.7.3.5.2 *Update*

Function Header:

```
uint8_t api_sha256_block_update (uint32_t *pblocks,
                                uint32_t nblocks,
                                uint32_t flags);
```

Description:

This routine programs hash engine with data address and the number of data blocks to process. Sha block must be initialized before this routine is called.

Inputs:

Input Parameters	Description
pblocks	An unsigned 32 bit integer pointer to data.
nblocks	An unsigned 32 bit integer specifying the number of 64 byte blocks
flags	An unsigned 32 bit integer specifying the following: <ul style="list-style-type: none"> • b[0] = 0/1 (do not start/start) • b[1] = 0/1 (if start, do not enable interrupts/ if start, enable interrupts)

Outputs:

The return values and their descriptions are presented below.

0 - Failure

1 - Success

4.7.3.5.3 *Implement Padding*

Function Header:

```
uint8_t api_sha256_block_finalize (uint32_t *ppadbuf,
```



```
uint32_t total_nblocks,
uint32_t flags);
```

Description:

This routine implement the standard SHA padding described in the FIPS standard.

Inputs:

Input Parameters	Description
ppadbuf	An unsigned 32 bit integer pointer to the buffer containing the final digest with padding.
total_nblocks	An unsigned 32 bit integer specifying the number of 64 byte blocks.
Flags	An unsigned 32 bit integer specifying the following: <ul style="list-style-type: none"> • b[0] = 0/1 (do not start/start) • b[1] = 0/1 (if start, do not enable interrupts/ if start, enable interrupts)

Outputs:

The return values and their descriptions are presented below.

0 - Failure

1 - Success

4.7.4 SHA512 Functions

4.7.4.1 Initialize

Function Header:

```
uint8_t api_sha512_init (SHA5_CONTEXT_T *sha5_ctx);
```

Description:

This function initializes the SHA5_CONTEXT_T data structure for SHA1.

Inputs:

Input Parameters	Description
Sha5_ctx	A pointer to the SHA5_CONTEXT_T data structure. The structure definition is given below: <pre>typedef struct sha5_context_s SHA5_CONTEXT_T; struct sha5_context_s { union { uint8_t b[SHA5_BLEN + 8]; uint32_t w[(SHA5_BLEN)/4 + 2]; } digest; uint32_t *pdigest; uint32_t *pblock; uint32_t mode; uint32_t block_len; uint16_t msg_blen[8]; // 128 bits uint8_t block[SHA5_BLOCK_BUF_LEN]; };</pre>

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_BAD_ADDR – This error is returned if the input points to NULL.

SHA_RET_OK – This value indicates a successful execution of requested operation.

Macro Name	Value
SHA_RET_OK	0
SHA_RET_ERR_BAD_ADDR	0x81

4.7.4.2 Update

Function Header:

```
uint8_t api_sha512_update (SHA5_CONTEXT_T * sha5_ctx,
                           const uint8_t *data,
                           uint32_t nbytes);
```

Description:

This routine programs hash engine with data address and the number of data blocks to process.

Inputs:

Input Parameters	Description
Sha5_ctx	A pointer to the SHA5_CONTEXT_T data structure. The structure definition is given below: <pre>typedef struct sha5_context_s SHA5_CONTEXT_T; struct sha5_context_s { union { uint8_t b[SHA5_BLEN + 8]; uint32_t w[(SHA5_BLEN)/4 + 2]; } digest; uint32_t *pdigest; uint32_t *pblock; uint32_t mode; uint32_t block_len; uint16_t msg_blen[8]; // 128 bits uint8_t block[SHA5_BLOCK_BUF_LEN]; };</pre>
data	An unsigned 8 bit pointer to the data.
nbytes	An unsigned 32 bit integer specifying the length of data (in bytes)

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_NULL_PTR– This error is returned if the input pointers point to NULL.

SHA_RET_OK – This value indicates a successful execution of requested operation.

SHA_RET_ERR_BUSY – this value is returned if the hash block is busy.

Macro Name	Value
SHA_RET_OK	0
SHA_RET_ERR_BUSY	0x80
SHA_RET_ERR_NULL_PTR	0x86

4.7.4.3 Implement Padding

Function Header:

```
uint8_t api_sha512_finalize (SHA5_CONTEXT_T * sha5_ctx);
```

Description:

This routine implements the standard SHA padding described in the FIPS standard.

Inputs:

Input Parameters	Description
Sha5_ctx	<p>A pointer to the SHA5_CONTEXT_T data structure. The structure definition is given below:</p> <pre>typedef struct sha5_context_s SHA5_CONTEXT_T; struct sha5_context_s { union { uint8_t b[SHA5_BLEN + 8]; uint32_t w[(SHA5_BLEN)/4 + 2]; } digest; uint32_t *pdigest; uint32_t *pblock; uint32_t mode; uint32_t block_len; uint16_t msg_blen[8]; // 128 bits uint8_t block[SHA5_BLOCK_BUF_LEN]; };</pre>

Outputs:

The return values and their descriptions are presented below.

SHA_RET_ERR_NULL_PTR – this value is returned when sha5_ctx points to NULL.

SHA_RET_ERR_BUSY – this value is returned if the hash block is busy.

SHA_RET_OK - This value indicates a successful execution of requested operation.

Macro Name	Value
SHA_RET_OK	0
SHA_RET_ERR_BUSY	0x80
SHA_RET_ERR_NULL_PTR	0x86

4.8 Miscellaneous ROM API

4.8.1 Check Version Number of the APIs

Function:

```
api_rom_ver
```

Function Header:

```
uint32_t api_rom_ver (void);
```

Description:

This routine returns the version number of the ROM API's.

Inputs:

None

Outputs:

The return value is an unsigned 32 bit integer reflecting the build information of ROM API's.

4.8.2 Load Firmware

Function:

api_ldr_api0

Function Header:

```
uint32_t api_ldr_api0 (uint32_t config,
                      LOAD_DESCR* pldr,
                      uint32_t* p256_ecdsa_pub,
                      uint32_t* p256_ecdh_prv,
                      uint32_t* buff2k);
```

Description:

This routine performs the firmware load process.

Inputs:

Input Parameters	Description
config	An unsigned 32 bit integer specifying the following: b[7:0] = interface 0 = Use ROM POR load interface 1 = Shared SPI 2 = Private SPI 3 = eSPI 4 = Internal SPI b[9:8] = SPI Freq MHz 0=48, 1=24, 2=16, 3=12 (N/A for eSPI) b[11:10] = SPI Drive Strength (N/A for eSPI) b[12] = SPI Slew Rate (N/A for eSPI) b[15:13] = 0 reserved b[19:16] = DMA channel (0-13) (N/A for eSPI) b[23:20] = 0 reserved b[30] = 0 Do not return to caller. 1 Return to caller b[31] = 0 ROM takes over interrupts (vector table set to ROM table) 1 caller retains ownership of interrupt vector table
pldr	A structure variable of type LOAD_DESCR <pre>typedef struct { uint32_t ld_addr; uint32_t byte_len; uint32_t spi_addr; uint32_t entry_addr; } LOAD_DESCR;</pre>
p256_ecdsa_pub	An unsigned 32 bit pointer to ecdsa public key , used in case firmware is authenticated
p256_ecdh_prv	An unsigned 32 bit pointer to ecdh private key , used in case firmware is encrypted.
buff2k	An unsigned 32 bit buffer required for implementing the function, aligned on a 16 byte boundary; minimum required size is 2k bytes.

Outputs:

The return value is an unsigned 32 bit integer and the descriptions are presented below.

ROM_API_LDR_BAD_PARAM – This error is returned when buff2k and pldr are NULL

ROM_API_LDR_ILLEGAL_IFACE – This error is returned when b[7:0] of config don't have the correct value for the interface.

ROM_API_LDR_BAD_FW_LEN – This error is returned when the byte_len parameter in pldr is not a multiple of 64 bytes.

ROM_API_LDR_FAIL – This error is returned when the loading fails

ROM_API_LDR_IFACE_INIT_ERR – This error is returned when the eSPI is not up and the flash channel is not ready

Macro Name	Value
ROM_API_LDR_BAD_PARAM	(0x80000001ul)
ROM_API_LDR_ILLEGAL_IFACE	(0x80000004ul)
ROM_API_LDR_BAD_FW_LEN	(0x80000007ul)
ROM_API_LDR_IFACE_INIT_ERR	(0x80000005ul)
ROM_API_LDR_FAIL	(0x80000006ul)

4.8.3 Apply GPIO Lock

Function header:

```
void api_rom_dis_lock_shd_spi(uint8_t lock_shd_spi);
```

Description:

Apply GPIO Locks as specified in customer section of EFUSE

Inputs:

Input parameters	Description
lock_shd_spi	An unsigned 8 bit value having the following definition: 0 (do not modify lock values) 1 (insure Shared SPI GPIO's are disabled (tri-state input) and these pins are locked).

Outputs: None

5 Appendix:

The symdef table for the B0 ROM API's

```
#<SYMDEFS># ARM Linker, 5.05 [Build 169]: Last Updated: Wed Jun 07 17:56:15 2017
```

```
0x00009028 D ec_sect571r1
0x00009044 D ec_sect409r1
0x00009060 D ec_sect283r1
0x0000907c D ec_sect233r1
0x00009098 D ec_sect163r2
0x000090d0 D ec_secp521r1
0x000090ec D ec_secp384r1
0x00009108 D ec_secp256r1
0x00009124 D ec_secp224r1
0x00009140 D ec_secp192r1
0x0000fb01 T api_rom_ver
0x0000fb05 T api_ldr_api0
0x0000fb09 T api_rom_dis_lock_shd_spi
0x0000fb0d T api_aes_hash_power
0x0000fb11 T api_aes_hash_reset
0x0000fb15 T api_aes_status
0x0000fb19 T api_aes_busy
0x0000fb1d T api_aes_is_done_status2
0x0000fb21 T api_aes_stop
0x0000fb25 T api_aes_start
0x0000fb29 T api_aes_prog_key
0x0000fb2d T api_aes_prog_iv
0x0000fb31 T api_aes_set_key
0x0000fb35 T api_aes_crypt
0x0000fb39 T api_aes_iclr
0x0000fb3d T api_rng_reset
0x0000fb41 T api_rng_power
0x0000fb45 T api_rng_is_on
0x0000fb49 T api_rng_start
0x0000fb4d T api_rng_stop
0x0000fb51 T api_rng_mode
0x0000fb55 T api_rng_get_num_random_words
0x0000fb59 T api_rng_get_random_bytes
0x0000fb5d T api_rng_get_random_words
0x0000fb61 T api_hash_start
0x0000fb65 T api_hash_busy
0x0000fb69 T api_hash_status
0x0000fb6d T api_hash_is_done_status
0x0000fb71 T api_hash_is_done_status2
0x0000fb75 T api_hash_iclr
0x0000fb79 T api_sha_direct_init
0x0000fb7d T api_sha_direct_update
0x0000fb81 T api_sha_direct_finalize
0x0000fb85 T api_sha1_init
0x0000fb89 T api_sha1_update
0x0000fb8d T api_sha1_finalize
0x0000fb91 T api_sha256_init
0x0000fb95 T api_sha256_update
0x0000fb99 T api_sha256_finalize
0x0000fb9d T api_sha256_block_init
0x0000fba1 T api_sha256_block_update
0x0000fba5 T api_sha256_block_finalize
```

Symdef continued...

```
0x0000fba9 T api_sha256_under56
0x0000fbad T api_sha512_init
0x0000fbb1 T api_sha512_update
0x0000fbb5 T api_sha512_finalize
0x0000fbb9 T api_pke_power
0x0000fbbd T api_pke_reset
0x0000fbc1 T api_pke_start
0x0000fbc5 T api_pke_status
0x0000fbc9 T api_pke_busy
0x0000fbcd T api_pke_is_done_status2
0x0000fbd1 T api_pke_ists_clear
0x0000fbd5 T api_pke_get_slot_addr
0x0000fbd9 T api_pke_fill_slot
0x0000fbdd T api_pke_clear_scm
0x0000fbe1 T api_pke_scm_clear_slot
0x0000fbe5 T api_pke_copy_from_scm
0x0000fbe9 T api_pke_copy_to_scm
0x0000fbef T api_pke_copy_to_scm2
0x0000fbf1 T api_pke_copy_to_scm4
0x0000fbf5 T api_pke_rsa_load_param
0x0000fbf9 T api_pke_rsa_get_param
0x0000fbfd T api_pke_rsa_operation
0x0000fc01 T api_pke_rsa_load_pub_key
0x0000fc05 T api_pke_rsa_load_prv_key
0x0000fc09 T api_pke_rsa_load_pub_keystruct
0x0000fc0d T api_pke_rsa_load_key
0x0000fc11 T api_pke_rsa_get_key
0x0000fc15 T api_pke_rsa_load crt_key
0x0000fc19 T api_pke_rsa_crypt
0x0000fc1d T api_pke_rsa crt_decrypt
0x0000fc21 T api_pke_ec_prog_curve
0x0000fc25 T api_pke_ec_point_double
0x0000fc29 T api_pke_ec_point_addition
0x0000fc2d T api_pke_ec_point_scalar_mult
0x0000fc31 T api_pke_ec_point_scalar_mult2
0x0000fc35 T api_pke_ec_point_scalar_mult3
0x0000fc39 T api_pke_ec_check_pxy_less_prime
0x0000fc3d T api_pke_ec_check_ab
0x0000fc41 T api_pke_ec_check_n
0x0000fc45 T api_pke_ec_check_poc2
0x0000fc49 T api_pke_ec_check_poc3
0x0000fc4d T api_pke_ec_kcda_keygen
0x0000fc51 T api_pke_ec_kcda_sign
0x0000fc55 T api_pke_ec_kcda_verify
0x0000fc59 T api_pke_srp_sc
0x0000fc5d T api_dma_dev_xfr_cfg
0x0000fc61 T api_qmspi_port_ctrl
0x0000fc65 T api_qmspi_port_drv_slew
0x0000fc69 T api_qmspi_is_done
0x0000fc6d T api_qmspi_is_done_status
0x0000fc71 T api_qmspi_init
0x0000fc75 T api_qmspi_soft_reset
0x0000fc79 T api_qmspi_flash_read24_dma
0x0000fc7d T api_qmspi_start
0x0000fc81 T api_qmspi_flash_cmd
0x0000fc85 T api_qmspi_flash_program_dma
```

Symdef continued...

```
0x0000fc89 T api_qmspi_ctx_init
0x0000fc8d T api_qmspi_ctx_freq_set
0x0000fc91 T api_qmspi_ctx_flash_rw_cmd
0x0000fc95 T api_qmspi_ctx_flash_rw_data_dma
0x0000fc99 T api_qmspi_ctx_flash_rd_data_blocking
0x0000fc9d T api_qmspi_ctx_start
0x0000fca1 T api_pke_ec25519_ptmult
0x0000fca5 T api_pke_ed25519_xrecov
0x0000fca9 T api_pke_ed25519_scalarmult
0x0000fcad T api_pke_ed25519_valid_sig
0x0000fcb1 T api_pke_modular_math
0x0000fcb5 T api_pke_ecdsa_verify3
0x0000fcb9 T api_pke_rsa_is_signature_valid
0x0000fcbd T api_pke_get_operand_slot
0x0000fcc1 T api_pke_set_operand_slot
0x0000fcc5 T api_pke_set_operand_slots
```


6 Revision History

Name	Revision Level	Date	Section	Remarks
Hemal Gujarathi – I15581	1.0	21 Apr 2016	4.3	Created a new copy to include changes for Bootrom A1 version
Akshaya Karthikeyan - I17306	1.1	13 May 2016	4.3	Added API for rsa crt decryption
Hemal Gujarathi – I15581	2.0	20 July 2016		Add new APIs
Arun Krishnan C21798	2.1	20 July 2016	4.5.1, 4.5.2, 4.5.3, 4.5.4, 4.5.6, 4.5.7, 4.5.18	Updated the following section for corrections Added Appendix for the symdef table used
Arun Krishnan C21798	2.2	22 July 2016	4.4.14	Added section 4.1.14 pke_write_scm32 Updated the Appendix for the symdef table used
Arun Krishnan C21798	2.3	22 July 2016	4.4.16	Updated the comment for the correct bit filed details
Akshaya Karthikeyan - I17306	2.4	17 August 2016	4.1,4.3,4.5,4.2.10	Added mode values for aes_crypt() Added note for block power enable, icct timer enable.
Akshaya Karthikeyan - I17306	2.5	06 September 2016	4.3,4.5,4.8	Added return code values for ECC, SHA block APIs.
Akshaya Karthikeyan - I17306	2.6	09 September 2016	4.9	Added section for miscellaneous ROM API - loader
Swastik Pramanik – I16169	2.7	22 November 2016	Section 4	Corrected API definitions as per BootROM A1. Also added the missing API definitions.
Swastik Pramanik – I16169	2.8	16 December 2016	Section 4.1, 4.6, 5	Removed qmspi_cfg_wrtie_cmd(), qmspi_write_dma() APIs and corrected the rng_power() API's description.
Swastik Pramanik – I16169	2.9	19 January 2017	Section 4.1	Removed qmspi_xmit_cmd() and added back qmspi_cfg_wrtie_cmd()..

Swastik Pramanik	2.10	6 April 2017	Section 4.6.9, Section 4.8.8, 4.8.9	Updated the rng_get_word() description, removed the instances of SHA384
Swastik Pramanik	2.11	22 June 2017	Section 4.2.8, Section 4.8.9, 4.8.10	Updated API descriptions as per customer reviews.
Manisha Mishra C19242	2.12	23 June 2017	Section 4, Section 4.2, Section 4.6, Appendix	Added a note in Section 4. Updated the AES functions as per BootROM B0 version. Updates the RNG functions as per BootROM B0 version. Updated the symdef table in the Appendix.
Manisha Mishra C19242	2.13	26 June 2017	Section 2.6, Section 4, Section 4.2, Section 4.6	Added Section 2.6 to include details about using the manual. Modified Note1 in Section 4. Updated the AES functions as per BootROM B0 version. Updated the RNG functions as per BootROM B0 version.
Manisha Mishra C19242	2.14	6 July 2017	Section 4.3, Section 4.4, Section 4.5	Updated the RSA functions as per BootROM B0 version. Updated the PKE functions as per BootROM B0 version. Updated the EC functions as per BootROM B0 version.
Manisha Mishra C19242	2.15	18 July 2017	Section 2.6 Section 4.8, Section 4.7,	Added details about deprecated APIs. Updated the SHA functions as per BootROM B0 version. Updated the Hash functions as per BootROM B0 version.
Manisha Mishra C19242	2.16	28 July 2017	Section 2.6 Section 4 Appendix Section 4.2 – 4.8 Section 1.4 Section 4.5.6 Section 4.5.1.3,4.3.4	Removed Section 2.6 Removed BootROM A1 API definitions from Section 4 Updated the symdef table in the Appendix. Modified the 'Output' sub heading for all APIs Added 'DMA' in Section 1.4 Modified the 'Inputs' for Section 4.5.6 Modified the 'Description' of the APIs in Section 4.5.1.3 and 4.3.4

Manisha Mishra C19242	2.17	28 July 2017	Section 4.9.2 Section 4.9.3	Changed Section 4.9.2 Removed Section 4.9.3
Manisha Mishra C19242	2.18	4 August 2017	Section 2.5 Section 4.7 Section 4.2.8 Section 4 Section 4.9 Section 4.6.4	Removed unnecessary details from Section 2.5 Added details about API api_hash_is_done_status2() and api_hash_is_done_status() Updated api_aes_iclr() Removed Note 3 in Section 4. Added details about api_rom_dis_lock_shd_spi() Modified Description in Section 4.6.4
Manisha Mishra C19242	2.19	9 August 2017	Section 4	Corrected input definitions of the APIs