



MPLAB Harmony Crypto Module Help

MPLAB Harmony Integrated Software Framework

Release Information for Crypto

Release Notes

This topic provides the release notes for this version of the MPLAB Harmony Crypto Library.

Description

MPLAB Harmony Version: 3.1.0 **Release Date:** November 2018

Software Requirements

Before using MPLAB Harmony, ensure that the following are installed:

- [MPLAB X IDE](#) 5.10
- [MPLAB XC32 C/C++ Compiler](#) V2.10
- MPLAB Harmony Configurator 3.0.0.44
- Harmony bsp repository, version 3.1.0
- Harmony core repository, version 3.1.0
- Harmony csp repository, version 3.1.0
- Harmony dev_packs repository, version 3.1.0
- Harmony mhc repository, version 3.1.0
- Harmony mplabx_plugin repository, version 3.0.0

What is New and Known Issues

The following tables list the features that have been changed or added and any known issues that have been identified. Any known issues that have yet to be resolved were retained from the previous release.

MPLAB Harmony Crypto:

Feature	Additions and Updates	Known Issues
Crypto Library	Initial Release of Crypto Library for Harmony 3. This includes support for the SAME70 family of MPCs.	<ul style="list-style-type: none">• Interactive help using the "Show User Manual Entry" in the Right-click menu for configuration options provided by this module is not yet available from within the MPLAB Harmony Configurator (MHC). Please see the "Configuring the Library" section in the help documentation in the doc folder for this module instead. Help is available in both CHM and PDF formats.

Applications

Large Hash Demonstration Application

The following demonstration describes the large hash functionality of the crypto module.

Description

This application demonstrates how to execute hashes on large blocks of data. In this case, the demonstration performs MD5, SHA-1, SHA-256, SHA-384, and SHA-512 hashing on a 512 * 1024 block of the letter 'a'.

After the hashing has been performed, the application outputs via the system console the results of the hashing, and the time it took to perform each form. It then compares the generated hashes with known values for each algorithm. If all tests pass a message is presented through the system console. If any tests fail a corresponding message is presented through the system console.

Building the Application

To build this project, the project corresponding to the functionality desired must be opened in MPLAB X IDE. The following table list and describes the project and supported configurations. The parent folder for these files is crypto/apps/large_hash/firmware.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

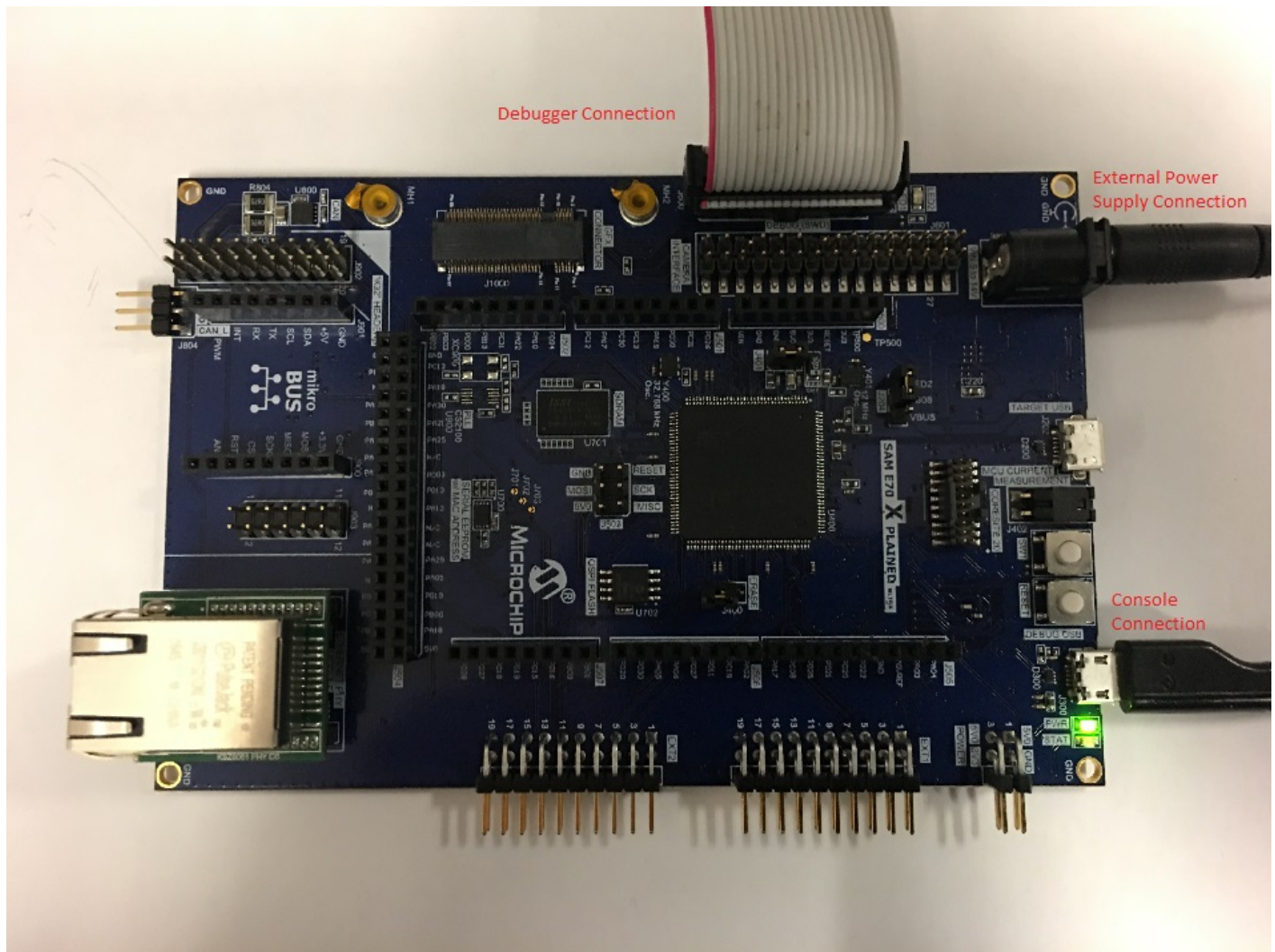
Project Name	BSP Used	Description
sam_e70_xplained_ultra.X	sam_e70_xplained_ultra	Large Hash project using the SAME70 and hardware accelerated cryptography
sam_e70_xplained_ultra_rtos.X	sam_e70_xplained_ultra	Large Hash project using the SAME70 and hardware accelerated cryptography. The software has been enabled to use freeRTOS.
sam_e70_xplained_ultra_sw.X	sam_e70_xplained_ultra	Large Hash project using the SAME70 and software cryptography

The application is built by using the standard MPLAB X IDE buttons.

Configuring the Hardware

SAM E79 Xplained Ultra Board

No hardware related configuration of jumper setting are necessary. The PC must be connected to the debug USB port to receive console output. An example is shown in the following figure.



Running the Demonstration

This demonstration exercises hashing functions on large blocks of data.

1. First connect the board to the PC as described in the '[Configuring the Hardware](#)' section.
2. Configure a terminal application (ex. Tera Term) to access the newly attached serial port. The parameters to use:
 - 115,200 bps
 - 8 data bits
 - no parity
 - 1 stop bit
 - no flow control
3. Compile and program the target device.
4. Observe the output from the serial console. An example is shown in the following figure.

```

COM4 - Tera Term VT
File Edit Setup Control Window Help
MD5 from feed: 30C2557E8302A5BEB290C71520D87F42 took 2650185
clock cycles
SHA from feed: F7FEC128D7FCD59222BA37368D3B7210D4C7B6EF took
1041150 clock cycles
SHA256 from feed: 85A84A75886E8A526DBEC4E16E3375FAA307B4AEAD79C9ED32
64C0477A6F6EBA took 934083 clock cycles
SHA384 from feed: A550561A6330048EFE826A97E5FED843FA1CE646A9BF546CCB
433C2FCB0E54821C4C945EED9A592B5BF43157E212F277 took 27769394 clock
cycles
SHA512 from feed: 7F49157FB359B39EA6DA934DC9A10709FEDF8846D139D0E637
A3C0FC833B6F42703858DBACEE28F4489B5E95FAB5E5655A25F838B0DC7BF3C84C7CC
0264F6A4F took 27488803 clock cycles
All tests passed.

```

Encrypt-Decrypt Demonstration Application

Demonstration of the encrypt/decrypt functionality of the crypto module.

Description

This demonstration exercises several cryptographic functions, including MD5, TDES, DES, AES, RSA, ECC, and Random Number Generation, to verify that the software or hardware is performing correctly.

Building the Application

To build this project, the project corresponding to the functionality desired must be opened in MPLAB X IDE. The following table lists and describes the project and supported configurations. The parent folder for these files is `crypto/apps/large_hash/firmware`.

MPLAB X IDE Project

This table lists the name and location of the MPLAB X IDE project folder for the demonstration.

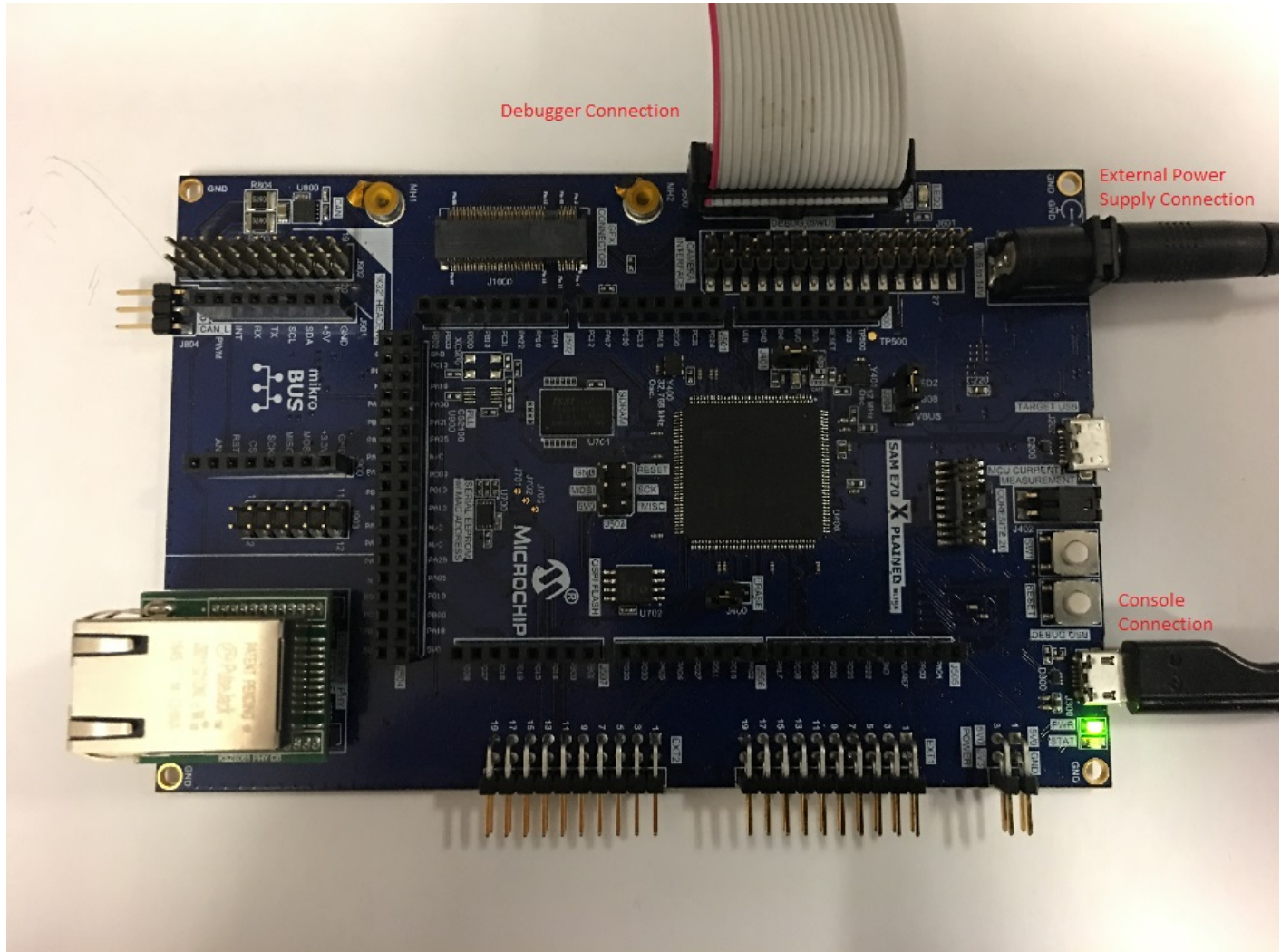
Project Name	BSP Used	Description
sam_e70_xplained_ultra.X	sam_e70_xplained_ultra	Encrypt/Decrypt project using the SAME70 and hardware accelerated cryptography
sam_e70_xplained_ultra_rtos.X	sam_e70_xplained_ultra	Encrypt/Decrypt project using the SAME70 and hardware accelerated cryptography. The software has been enabled to use freeRTOS.
sam_e70_xplained_ultra_sw.X	sam_e70_xplained_ultra	Encrypt/Decrypt project using the SAME70 and software cryptography

The application is built by using the standard MPLAB X IDE buttons.

Configuring the Hardware

SAM E79 Xplained Ultra Board

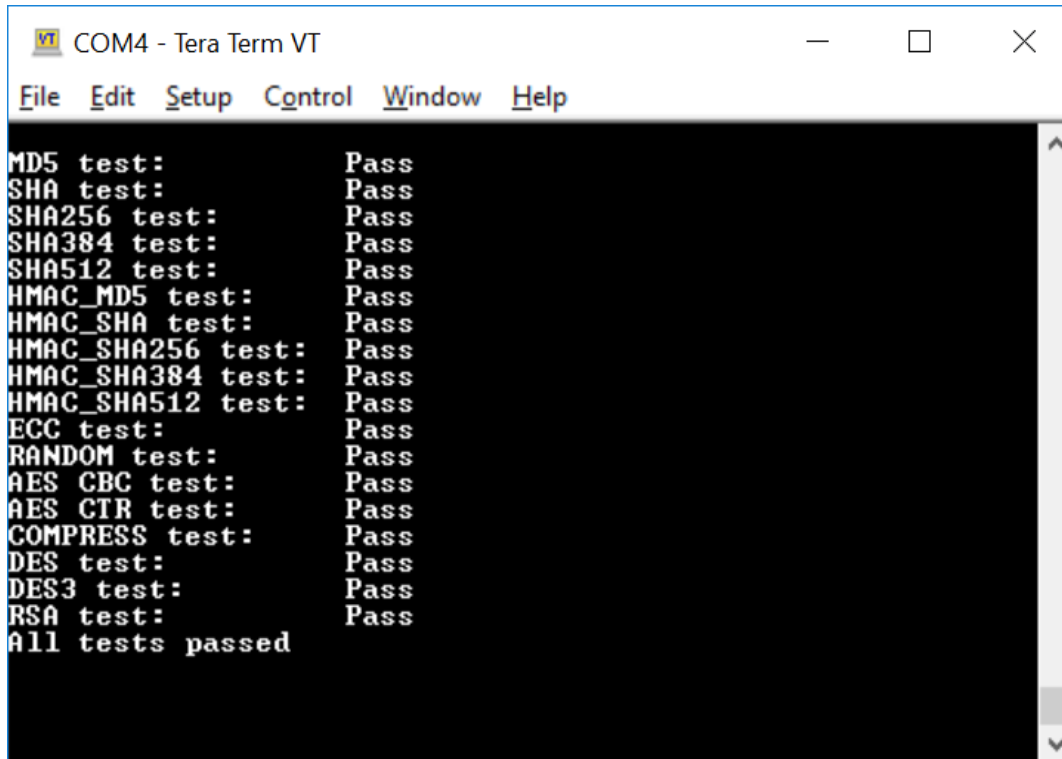
No hardware related configuration of jumper setting are necessary. The PC must be connected to the debug USB port to receive console output. An example is shown in the following figure.



Running the Demostration

This demonstration exercises various encryption, decryption, hashing and random number functions.

1. First connect the board to the PC as described in the '[Configuring the Hardware](#)' section.
2. Configure a terminal application (ex. Tera Term) to access the newly attached serial port. The parameters to use:
 - 115,200 bps
 - 8 data bits
 - no parity
 - 1 stop bit
 - no flow control
3. Compile and program the target device.
4. Observe the output from the serial console, it should display as follows.



The image shows a screenshot of a Tera Term VT window titled "COM4 - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The main area displays a list of cryptographic tests and their results, all of which are "Pass". The tests listed are: MD5 test, SHA test, SHA256 test, SHA384 test, SHA512 test, HMAC_MD5 test, HMAC_SHA test, HMAC_SHA256 test, HMAC_SHA384 test, HMAC_SHA512 test, ECC test, RANDOM test, AES CBC test, AES CTR test, COMPRESS test, DES test, DES3 test, and RSA test. The final line of the output is "All tests passed".

```
COM4 - Tera Term VT
File Edit Setup Control Window Help
MD5 test: Pass
SHA test: Pass
SHA256 test: Pass
SHA384 test: Pass
SHA512 test: Pass
HMAC_MD5 test: Pass
HMAC_SHA test: Pass
HMAC_SHA256 test: Pass
HMAC_SHA384 test: Pass
HMAC_SHA512 test: Pass
ECC test: Pass
RANDOM test: Pass
AES CBC test: Pass
AES CTR test: Pass
COMPRESS test: Pass
DES test: Pass
DES3 test: Pass
RSA test: Pass
All tests passed
```

Crypto Library Help

This section describes the Cryptographic (Crypto) Library that is available in MPLAB Harmony.

Introduction

This library provides a software Cryptographic Library that is available on the Microchip family of microcontrollers with a convenient C language interface.

Description

The Cryptographic Library includes functions to perform encryption, decryption, hashing, authentication, and compression within the embedded application. Random number generation (RNG) functions are also provided.

Block Ciphers

The library provides DES, 3DES, and AES for block cipher needs. Depending on the algorithm in use, CBC and CTR modes are supported.

Public Key Cryptography

The library provides RSA and Elliptic Curve Cryptography (ECC) for Public Key Cryptography, and Diffie-Hellman (DH) for key agreement arrangements.

Hash Functions

The library provides MD5, SHA, SHA-256, SHA-384, and SHA-512 for hashing. These functions do not require keys or initialization vectors (IV).

Random Number Generation Functions

The library provides functions to generate either a single pseudo-random number, or a block of such numbers.

Using the Library

This topic describes the basic architecture of the Cryptographic (Crypto) Library and provides information and examples on its use.

Description

Interface Header File: [crypto.h](#)

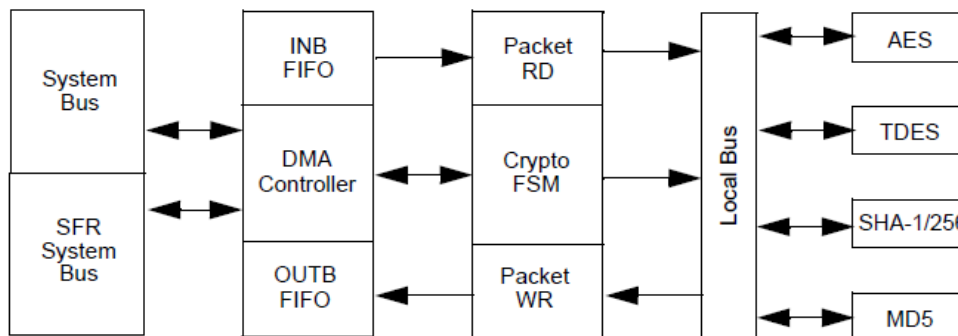
The interface to the Crypto Library is defined in the [crypto.h](#) header file. Any C language source (.c) file that uses the Crypto Library should include [crypto.h](#).

Abstraction Model

This library provides the low-level abstraction of the Cryptographic Library module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

Cryptographic (Crypto) Software Abstraction Block Diagram



Library Overview

The [Library Interface](#) routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Cryptographic (Crypto) Library module.

Library Interface Section	Description
General Functions	Provides an error string function, which takes an error and converts it into human-readable text.
Compression Functions	Provides Huffman compression and decompression functions.
MD5 Functions	Provides data add, finalize, and initialize MD5 functions.
Random Number Generator Functions	Provides get, initialize, and block generate RNG functions.
AES Encryption/Decryption Functions	Provides AES encryption and decryption functions.
ECC Encryption/Decryption Functions	Provides ECC encryption and decryption functions.
RSA Encryption/Decryption Functions	Provides RSA encryption and decryption functions.
Triple DES Encryption/Decryption Functions	Provides 3DES encryption and decryption functions.
HMAC Hash Functions	Provides HMAC data add, finalize, and set key Hash functions.
SHA Hash Functions	Provides SHA data add, finalize, and initialize Hash functions.
SHA256 Hash Functions	Provides SHA256 data add, finalize, and initialize Hash functions.
SHA384 Hash Functions	Provides SHA384 data add, finalize, and initialize Hash functions.
SHA512 Hash Functions	Provides SHA512 data add, finalize, and initialize Hash functions.

Configuring the Library

The configuration of the Cryptographic Library is based on the file configuration .h.

This header file contains the configuration selection for the Cryptographic Library. Based on the selections made, the Cryptographic Library may support the selected features. These configuration settings will apply to all instances of the Cryptographic Library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build.

Building the Library

This section lists the files that are available in the Crypto Library.

Description

The following lists and describes the header (.h) and source (.c) files that implement this library. The parent folder for these files is <harmony 3 root>/crypto.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using #include) by any code that uses this library.

Source File Name	Description
crypto.h	Includes all MPLAB Harmony-compatible function calls for the Crypto Library.

Required File(s)



All of the required files are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

Optional File(s)


MHC

All of the optional files are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

Module Dependencies

The Crypto Library does not depend on any other modules.

Library Interface

a) General Functions

	Name	Description
	CRYPT_ERROR_StringGet	Reports the nature of an error.

b) Compression Functions

	Name	Description
	CRYPT_HUFFMAN_Compress	Compresses a block of data.
	CRYPT_HUFFMAN_DeCompress	Decompresses a block of data.

c) MD5 Functions

	Name	Description
	CRYPT_MD5_DataAdd	Updates the hash with the data provided.
	CRYPT_MD5_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_MD5_Initialize	Initializes the internal structures necessary for MD5 hash calculations.
	CRYPT_MD5_DataSizeSet	This function sets the size of the input data for use with hardware accelerated encryption.

d) Random Number Generator Functions




	Name	Description
	CRYPT_RNG_Initialize	Initializes the random number generator.
	CRYPT_RNG_BlockGenerate	Createa several random numbers.
	CRYPT_RNG_Get	Gets one random number.

e) AES Encryption/Decryption Functions








	Name	Description
	CRYPT_AES_CBC_Decrypt	Performs AES decryption using Cipher-Block-Chaining (CBC).
	CRYPT_AES_CBC_Encrypt	Performs AES encryption using Cipher-Block-Chaining (CBC).
	CRYPT_AES_CTR_Encrypt	Performs AES encryption using Counter (CTR).
	CRYPT_AES_DIRECT_Decrypt	Directs decryption of one block of data.
	CRYPT_AES_DIRECT_Encrypt	Directs encryption of one block of data.
	CRYPT_AES_IvSet	Sets the initialization vector (IV) for AES processing.
	CRYPT_AES_KeySet	Sets the key and initialization vector (IV) for AES processing.

f) ECC Encryption/Decryption Functions





	Name	Description
	CRYPT_ECC_DHE_KeyMake	Creates a new ECC key.
	CRYPT_ECC_DHE_SharedSecretMake	Creates an ECC shared secret between two keys.
	CRYPT_ECC_DSA_HashSign	Signs a message digest.
	CRYPT_ECC_DSA_HashVerify	Verifies an ECC signature.
	CRYPT_ECC_Free	Cleans up an Elliptic Curve Cryptography (ECC) Context.
	CRYPT_ECC_Initialize	Initializes the context for Elliptic Curve Cryptography (ECC).
	CRYPT_ECC_KeySizeGet	Returns the key size in octets.

	CRYPT_ECC_PrivateImport	Imports private key pair in X9.63 format.
	CRYPT_ECC_PublicExport	Exports public ECC key in ANSI X9.63 format.
	CRYPT_ECC_PublicImport	Imports public key in ANSI X9.63 format.
	CRYPT_ECC_SignatureSizeGet	Returns the signature size in octets.




g) RSA Encryption/Decryption Functions

	Name	Description
	CRYPT_RSA_EncryptSizeGet	Gets the size of the RSA Key.
	CRYPT_RSA_Free	Releases the memory used for the key and cleans up the context.
	CRYPT_RSA_Initialize	Initializes the internal structures necessary for RSA processing.
	CRYPT_RSA_PrivateDecrypt	Decrypts data using a private key.
	CRYPT_RSA_PrivateKeyDecode	Constructs the Private Key from a DER certificate.
	CRYPT_RSA_PublicEncrypt	Encrypts data using a public key.
	CRYPT_RSA_PublicKeyDecode	Constructs the Public Key from a DER certificate.





h) Triple DES (3DES) Encryption/Decryption Functions

	Name	Description
	CRYPT_TDES_CBC_Decrypt	Decrypts a data block using Triple DES.
	CRYPT_TDES_CBC_Encrypt	Encrypts a data block using Triple DES.
	CRYPT_TDES_IvSet	Sets the Initialization Vector (IV) for a Triple DES operation.
	CRYPT_TDES_KeySet	Initialization of Triple DES context.





i) HMAC Hash Functions

	Name	Description
	CRYPT_HMAC_DataAdd	Adds data to the HMAC calculation.
	CRYPT_HMAC_Finalize	Completes the HMAC calculation and get the results.
	CRYPT_HMAC_SetKey	Initializes the HMAC context and set the key for the hash.




j) SHA Hash functions

	Name	Description
	CRYPT_SHA_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA_Initialize	Initializes the internal structures necessary for SHA hash calculations.
	CRYPT_SHA_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.



k) SHA256 Hash Functions


	Name	Description
	CRYPT_SHA256_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA256_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA256_Initialize	Initializes the internal structures necessary for SHA256 hash calculations.
	CRYPT_SHA256_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.

l) SHA384 Hash Functions












	Name	Description
	CRYPT_SHA384_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA384_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA384_Initialize	Initializes the internal structures necessary for SHA384 hash calculations.

m) SHA512 Hash Functions

	Name	Description
	CRYPT_SHA512_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA512_Finalize	Finalizes the hash and puts the result into digest.

	CRYPT_SHA512_Initialize	Initializes the internal structures necessary for SHA512 hash calculations.
---	---	---

n) Data Types and Constants

	Name	Description
	CRYPT_AES_CTX	AES
	CRYPT_ECC_CTX	ECC
	CRYPT_HMAC_CTX	HMAC
	CRYPT_MD5_CTX	MD5
	CRYPT_RNG_CTX	RNG
	CRYPT_RSA_CTX	RSA
	CRYPT_SHA_CTX	SHA
	CRYPT_SHA256_CTX	SHA-256
	CRYPT_SHA384_CTX	SHA-384
	CRYPT_SHA512_CTX	SHA-512
	CRYPT_TDES_CTX	TDES
	MC_CRYPT_API_H	Defines Microchip CRYPTO API layer

Description

This section describes the Application Programming Interface (API) functions of the Cryptographic Library.
Refer to each section for a detailed description.

a) General Functions

CRYPT_ERROR_StringGet Function

Reports the nature of an error.

File

[crypto.h](#)

C

```
int CRYPT_ERROR_StringGet(int, char*);
```

Returns

- BAD_FUNC_ARG - A null string was passed for the return message.
- 0 - A null string was not passed for the return message.

Description

This function takes an error and converts it into human-readable text.

Remarks

String needs to be ≥ 80 chars.

Preconditions

None.

Example

```
char msg[80];

CRYPT_ERR_StringGet(ret, msg);
```

Parameters

Parameters	Description
int	Error code to convert.
str	Pointer to buffer to store the message. Must hold at least 80 characters.

Function

```
int CRYPT_ERROR_StringGet(int err, char* str)
```

b) Compression Functions

CRYPT_HUFFMAN_Compress Function

Compresses a block of data.

File

[crypto.h](#)

C

```
int CRYPT_HUFFMAN_Compress(unsigned char*, unsigned int, const unsigned char*, unsigned int,  
unsigned int);
```

Returns

- negative - error code
- positive - bytes stored in out buffer

Description

This function compresses a block of data using Huffman encoding.

Remarks

Output buffer must be large enough to hold the contents of the operation.

Preconditions

None.

Example

```
const unsigned char text[] = "...";  
unsigned int inSz = sizeof(text);  
unsigned int outSz;  
unsigned char cBuffer[1024];  
  
int ret;  
  
ret = CRYPT_HUFFMAN_COMPRESS(cBuffer, sizeof(cBuffer), text, inSz, 0);
```

Parameters

Parameters	Description
out	Pointer to location to store the compressed data.
outSz	Maximum size of the output data in bytes.
in	Point to location of source data.
inSz	Size of the input data in bytes.
flags	Flags to control how compress operates

Function

int CRYPT_HUFFMAN_Compress(unsigned char* out, unsigned int outSz, const unsigned char* in, unsigned int inSz, unsigned int flags)

CRYPT_HUFFMAN_DeCompress Function

Decompresses a block of data.

File

[crypto.h](#)

C

```
int CRYPT_HUFFMAN_DeCompress(unsigned char*, unsigned int, const unsigned char*, unsigned int);
```

Returns

- negative - Error code
- positive - Bytes stored in out buffer

Description

This function decompresses a block of data using Huffman encoding.

Remarks

Output buffer must be large enough to hold the contents of the operation.

Preconditions

None.

Example

```
unsigned char cBuffer[1024];
unsigned char dBuffer[1024];

int ret

ret = CRYPT_HUFFMAN_DeCompress(dBuffer, sizeof(dBuffer), cBuffer, msglen);
```

Parameters

Parameters	Description
out	Pointer to destination buffer
outSz	Size of destination buffer
in	Pointer to source buffer to decompress
inSz	Size of source buffer to decompress

Function

int CRYPT_HUFFMAN_DeCompress(unsigned char* out, unsigned int outSz, const unsigned char* in, unsigned int inSz)

c) MD5 Functions

CRYPT_MD5_DataAdd Function

Updates the hash with the data provided.

File

[crypto.h](#)

C

```
int CRYPT_MD5_DataAdd(CRYPT_MD5_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in md5 or input
- 0 - An invalid pointer was not passed to the function

Description

This function updates the hash with the data provided.

Remarks

To preserve the validity of the MD5 hash, nothing must modify the context holding variable between calls to CRYPT_MD5_DataAdd.

Preconditions

The MD5 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

```
int CRYPT_MD5_DataAdd( CRYPT_MD5_CTX* md5, const unsigned char* input, unsigned int sz)
```

CRYPT_MD5_Finalize Function

Finalizes the hash and puts the result into digest.

File

[crypto.h](#)

C

```
int CRYPT_MD5_Finalize(CRYPT_MD5_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in md5 or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the MD5 hash, nothing must modify the context holding variable between calls to [CRYPT_MD5_DataAdd](#) and [CRYPT_MD5_Finalize](#).

Preconditions

The MD5 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

int [CRYPT_MD5_Finalize](#)([CRYPT_MD5_CTX](#)* md5, unsigned char* digest)

CRYPT_MD5_Initialize Function

Initializes the internal structures necessary for MD5 hash calculations.

File

[crypto.h](#)

C

```
int CRYPT\_MD5\_Initialize(CRYPT_MD5_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for MD5 hash calculations.

Remarks

All MD5 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_MD5_CTX md5;
uint8_t buffer[1024];
uint8_t md5sum[MD5_DIGEST_SIZE];

CRYPT_MD5_Initialize(&md5);
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.

Function

```
int CRYPT_MD5_Initialize( CRYPT\_MD5\_CTX\* md5)
```

CRYPT_MD5_DataSizeSet Function

This function sets the size of the input data for use with hardware accelerated encryption.

File

[crypto.h](#)

C

```
int CRYPT_MD5_DataSizeSet(CRYPT\_MD5\_CTX\*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

The hardware encryption module needs to know the size of the data before it starts processing. This function sets that value.

Remarks

All MD5 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_MD5_CTX md5;  
uint8_t buffer[1024];  
uint8_t md5sum[MD5_DIGEST_SIZE];  
  
CRYPT_MD5_Initialize(&md5);  
CRYPT_MD5DataSizeSet(&md5, sizeof(buffer));  
CRYPT_MD5_DataAdd(&md5, buffer, sizeof(buffer));  
CRYPT_MD5_Finalize(&md5, md5sum);
```

Parameters

Parameters	Description
md5	Pointer to CRYPT_MD5_CTX structure which holds the hash values.
msgSize	Size of the data (in bytes) that will be processed.

Function

```
int CRYPT_MD5_DataSizeSet( CRYPT\_MD5\_CTX\* md5, unsigned int msgSize)
```

d) Random Number Generator Functions

CRYPT_RNG_Initialize Function

Initializes the random number generator.

File

[crypto.h](#)

C

```
int CRYPT_RNG_Initialize(CRYPT_RNG_CTX*);
```

Returns

- negative - An error occurred setting up the random number generator.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the context that stores information relative to random number generation.

Preconditions

None.

Example

```
#define RANDOM_BYTE_SZ 32

int          ret;
CRYPT_RNG_CTX mcRng;
byte         out[RANDOM_BYTE_SZ];

ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RNG_Get(&mcRng, &out[0]);

ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

Parameters

Parameters	Description
rng	Pointer to random number generator context.

Function

```
int CRYPT_RNG_Initialize( CRYPT_RNG_CTX* rng)
```

CRYPT_RNG_BlockGenerate Function

Createa several random numbers.

File

[crypto.h](#)

C

```
int CRYPT_RNG_BlockGenerate(CRYPT_RNG_CTX*, unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function generates several random numbers and places them in the space allocated.

Preconditions

RNG context was initialized using the [CRYPT_RNG_Initialize](#) function.

Example

```
#define RANDOM_BYTE_SZ 32

int          ret;
CRYPT_RNG_CTX mcRng;
byte         out[RANDOM_BYTE_SZ];

ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RNG_Get(&mcRng, &out[0]);

ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

Parameters

Parameters	Description
rng	Pointer to context which saves state between calls.
b	Pointer to buffer to store the random numbers.
sz	Number of 8-bit random numbers to generate.

Function

int CRYPT_RNG_BlockGenerate([CRYPT_RNG_CTX*](#) rng, unsigned char* b, unsigned int sz)

CRYPT_RNG_Get Function

Gets one random number.

File

[crypto.h](#)

C

```
int CRYPT_RNG_Get(CRYPT_RNG_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- Less than 0 - An error occurred.
- 0 or greater - Success.

Description

This function gets one random number from the random number generator.

Preconditions

RNG context was initialized using the [CRYPT_RNG_Initialize](#) function.

Example

```
#define RANDOM_BYTE_SZ 32

int          ret;
CRYPT_RNG_CTX mcRng;
byte         out[RANDOM_BYTE_SZ];

ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RNG_Get(&mcRng, &out[0]);

ret = CRYPT_RNG_BlockGenerate(&mcRng, out, RANDOM_BYTE_SZ);
```

Parameters

Parameters	Description
rng	Pointer to context which saves state between calls.
b	Pointer to 8-bit location to store the result.

Function

```
int CRYPT_RNG_Get( CRYPT_RNG_CTX* rng, unsigned char* b)
```

e) AES Encryption/Decryption Functions

CRYPT_AES_CBC_Decrypt Function

Performs AES decryption using Cipher-Block-Chaining (CBC).

File

[crypto.h](#)

C

```
int CRYPT_AES_CBC_Decrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function decrypts a block of data using the AES algorithm in Cipher- Block-Chaining (CBC) mode.

Remarks

The output buffer must be equal in size to the input buffer.

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to [CRYPT_AES_KeySet](#) and [CRYPT_AES_IvSet](#).

Example

```
CRYPT_AES_CTX mcAes;
int          ret;
byte         out1[AES_TEST_SIZE];
byte         out2[AES_TEST_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuvwxyz", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_CBC_Decrypt(&mcAes, out2, out1, AES_TEST_SIZE);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the decryption pass.
in	Pointer to buffer holding the data to be decrypted.
inSz	Size of the input data, in bytes.

Function

```
int CRYPT_AES_CBC_Decrypt( CRYPT_AES_CTX* aes, unsigned char* out,  
const unsigned char* in, unsigned int inSz)
```

CRYPT_AES_CBC_Encrypt Function

Performs AES encryption using Cipher-Block-Chaining (CBC).

File

[crypto.h](#)

C

```
int CRYPT_AES_CBC_Encrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts a block of data using the AES algorithm in Cipher- Block-Chaining (CBC) mode.

Remarks

The output buffer must be equal in size to the input buffer.

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to [CRYPT_AES_KeySet](#) and [CRYPT_AES_IvSet](#).

Example

```
CRYPT_AES_CTX mcAes;  
int          ret;  
byte         out1[AES_TEST_SIZE];  
  
strncpy((char*)key, "1234567890abcdefghijklmnopqrstuv", 32);  
strncpy((char*)iv, "1234567890abcdef", 16);  
  
ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);  
ret = CRYPT_AES_CBC_Encrypt(&mcAes, out1, ourData, AES_TEST_SIZE);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the encryption pass.
in	Pointer to buffer holding the data to be encrypted.
inSz	Size of the input data, in bytes.

Function

```
int CRYPT_AES_CBC_Encrypt( CRYPT_AES_CTX* aes, unsigned char* out,  
const unsigned char* in, unsigned int inSz)
```

CRYPT_AES_CTR_Encrypt Function

Performs AES encryption using Counter (CTR).

File

[crypto.h](#)

C

```
int CRYPT_AES_CTR_Encrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts a block of data using the AES algorithm in Counter (CTR) mode.

Remarks

The output buffer must be equal in size to the input buffer.

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to [CRYPT_AES_KeySet](#) and [CRYPT_AES_IvSet](#).

Example

```
CRYPT_AES_CTX mcAes;
int          ret;
byte         out1[AES_TEST_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuvwxyz", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_CTR_Encrypt(&mcAes, out1, ourData, AES_TEST_SIZE);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the encryption pass.
in	Pointer to buffer holding the data to be encrypted.
inSz	Size of the input data, in bytes.

Function

```
int CRYPT_AES_CTR_Encrypt( CRYPT_AES_CTX* aes, unsigned char* out,
const unsigned char* in, unsigned int inSz)
```

CRYPT_AES_DIRECT_Decrypt Function

Directs decryption of one block of data.

File

[crypto.h](#)

C

```
int CRYPT_AES_DIRECT_Decrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function decrypts one block of data, equal to the AES block size.

Remarks

Input and output buffers must be equal in size (CRYPT_AES_BLOCK_SIZE).

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to [CRYPT_AES_KeySet](#) and [CRYPT_AES_IvSet](#).

Example

```
CRYPT_AES_CTX mcAes;
int          ret;
byte         out1[CRYPT_AES_BLOCK_SIZE];
byte         out2[CRYPT_AES_BLOCK_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuvwxyz", 32);
strncpy((char*)iv, "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_DIRECT_Decrypt(&mcAes, out2, out1);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the decryption.
in	Pointer to buffer where the data to decrypt is located.

Function

```
int CRYPT_AES_DIRECT_Decrypt( CRYPT_AES_CTX*, unsigned char*,
const unsigned char*)
```

CRYPT_AES_DIRECT_Encrypt Function

Directs encryption of one block of data.

File

[crypto.h](#)

C

```
int CRYPT_AES_DIRECT_Encrypt(CRYPT_AES_CTX*, unsigned char*, const unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts one block of data, equal to the AES block size.

Remarks

Input and output buffers must be equal in size (CRYPT_AES_BLOCK_SIZE).

Preconditions

Key and Initialization Vector (IV) must be set earlier with a call to [CRYPT_AES_KeySet](#) and [CRYPT_AES_IvSet](#).

Example

```
CRYPT_AES_CTX mcAes;
int          ret;
byte         out1[CRYPT_AES_BLOCK_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuvwxyz", 32);
```

```

strncpy((char*)iv, "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
ret = CRYPT_AES_DIRECT_Encrypt(&mcAes, out1, ourData);

```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
out	Pointer to buffer to store the results of the encryption.
in	Pointer to buffer where the data to encrypt is located.

Function

```

int CRYPT_AES_DIRECT_Encrypt( CRYPT_AES_CTX* aes, unsigned char* out,
const unsigned char* in)

```

CRYPT_AES_IvSet Function

Sets the initialization vector (IV) for AES processing.

File

[crypto.h](#)

C

```

int CRYPT_AES_IvSet(CRYPT_AES_CTX*, const unsigned char*);

```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function sets the IV that AES will use for later processing.

Remarks

Direction, as set previously in [CRYPT_AES_KeySet](#), is preserved.

Preconditions

The key must be set previously with [CRYPT_AES_KeySet](#).

Example

```

CRYPT_AES_CTX mcAes;
int          ret;

strncpy((char*)iv, "1234567890abcdef", 16);

ret = CRYPT_AES_IvSet(&mcAes, iv);

```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
iv	Pointer to buffer holding the initialization vector.

Function

```

int CRYPT_AES_IvSet( CRYPT_AES_CTX* aes, const unsigned char* iv)

```

CRYPT_AES_KeySet Function

Sets the key and initialization vector (IV) for AES processing.

File

[crypto.h](#)

C

```
int CRYPT_AES_KeySet(CRYPT_AES_CTX*, const unsigned char*, unsigned int, const unsigned char*, int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function sets the key and IV, and the direction (encryption or decryption) that AES will later perform.

Preconditions

None.

Example

```
CRYPT_AES_CTX mcAes;
int          ret;

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuvwxyz", 32);
strncpy((char*)iv,  "1234567890abcdef", 16);

ret = CRYPT_AES_KeySet(&mcAes, key, 16, iv, CRYPT_AES_ENCRYPTION);
```

Parameters

Parameters	Description
aes	Pointer to context which saves state between calls.
key	Pointer to buffer holding the key itself.
keyLen	Length of key in bytes.
iv	Pointer to buffer holding the initialization vector.
dir	Which operation (CRYPT_AES_ENCRYPTION or CRYPT_AES_DECRYPTION).

Function

```
int CRYPT_AES_KeySet( CRYPT_AES_CTX* aes, const unsigned char* key,
unsigned int keylen, const unsigned char* iv, int dir)
```

f) ECC Encryption/Decryption Functions

CRYPT_ECC_DHE_KeyMake Function

Creates a new ECC key.

File

[crypto.h](#)

C

```
int CRYPT_ECC_DHE_KeyMake(CRYPT_ECC_CTX*, CRYPT_RNG_CTX*, int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Could not create the memory buffer for the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function creates a new ECC key.

Preconditions

The context must have been initialized with a call to [CRYPT_ECC_Initialize](#). The random number generator context must have been initialized with a call to [CRYPT_RNG_Initialize](#).

Example

```
CRYPT_ECC_CTX userA;  
int          ret;  
byte         sharedA[100];  
unsigned int  aSz = (unsigned int)sizeof(sharedA);  
unsigned int  usedA = 0;  
  
ret = CRYPT_ECC_Initialize(&userA);  
ret = CRYPT_ECC_DHE_KeyMake(&userA, &mcRng, 32);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
rng	Pointer to the context for the random number generator.
keySz	The size of the key desired.

Function

```
int CRYPT_ECC_DHE_KeyMake( CRYPT_ECC_CTX*, CRYPT_RNG_CTX*, int)
```

CRYPT_ECC_DHE_SharedSecretMake Function

Creates an ECC shared secret between two keys.

File

[crypto.h](#)

C

```
int CRYPT_ECC_DHE_SharedSecretMake(CRYPT_ECC_CTX*, CRYPT_ECC_CTX*, unsigned char*, unsigned  
int, unsigned int*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Could not create the memory buffer for the shared secret.
- 0 - An invalid pointer was not passed to the function.

Description

This function takes two ECC contexts (one public, one private) and creates a shared secret between the two. The secret conforms to EC-DH from ANSI X9.63.

Preconditions

Both contexts must have been initialized with a call to [CRYPT_ECC_Initialize](#). Both contexts have had their respective keys

imported or created.

Example

```
CRYPT_ECC_CTX userA;
CRYPT_ECC_CTX userB;
int          ret;
byte        sharedA[100];
unsigned int aSz  = (unsigned int)sizeof(sharedA);
unsigned int usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
ret = CRYPT_ECC_Initialize(&userB);
...
// Make or import the appropriate keys
...
ret = CRYPT_ECC_DHE_SharedSecretMake(&userA, &userB, sharedA, aSz, &usedA);
```

Parameters

Parameters	Description
priv	Pointer to the private ECC context (with the private key).
pub	Pointer to the public ECC context (with the public key).
out	Destination of the shared secret.
outSz	The max size of the shared secret.
usedSz	Resulting size of the shared secret.

Function

int CRYPT_ECC_DHE_SharedSecretMake(CRYPT_ECC_CTX* priv, CRYPT_ECC_CTX* pub,
unsigned char* out, unsigned int outSz, unsigned int* usedSz)

CRYPT_ECC_DSA_HashSign Function

Signs a message digest.

File

[crypto.h](#)

C

```
int CRYPT_ECC_DSA_HashSign(CRYPT_ECC_CTX*, CRYPT_RNG_CTX*, unsigned char*, unsigned int,  
unsigned int*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function takes a message digest and signs it using a private ECC key.

Preconditions

The ECC context must have been initialized with a call to [CRYPT_ECC_Initialize](#). The RNG context must have been initialized with a call to [CRYPT_RNG_Initialize](#). The private key used for the signature must have been imported or created prior to calling this function.

Example

```
CRYPT_ECC_CTX userA;
int          ret;
byte        sig[100];
unsigned int sigSz = (unsigned int)sizeof(sig);
unsigned int usedA = 0;
```

```

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_DSA_HashSign(&userA, &mcRng, sig, sigSz, &usedA, ourData,
                             CRYPT_SHA_DIGEST_SIZE);

```

Parameters

Parameters	Description
ecc	Pointer to ECC context which saves state between calls and holds keys.
rng	Pointer to Random Number Generator context.
sig	Destination for the signature.
sigSz	The max size of the signature, in bytes.
usedSz	The resulting size of the signature, in bytes.
in	Pointer to the message digest to sign.
inSz	The length of the digest, in bytes.

Function

int CRYPT_ECC_DSA_HashSign([CRYPT_ECC_CTX*](#) ecc, [CRYPT_RNG_CTX*](#) rng, unsigned char* sig, unsigned int sigSz, unsigned int* usedSz, const unsigned char* in, unsigned int inSz)

CRYPT_ECC_DSA_HashVerify Function

Verifies an ECC signature.

File

[crypto.h](#)

C

```

int CRYPT_ECC_DSA_HashVerify(CRYPT_ECC_CTX*, const unsigned char*, unsigned int, unsigned
char*, unsigned int, int*);

```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Memory could not be allocated for the operation.
- 0 - An invalid pointer was not passed to the function.

Description

This function verifies that an ECC signature is valid.

Preconditions

The ECC context must have been initialized with a call to [CRYPT_ECC_Initialize](#). The key used for the signature must have been imported or created prior to calling this function.

Example

```

CRYPT_ECC_CTX userA;
int          ret;
byte        sig[100];
unsigned int sigSz = (unsigned int)sizeof(sig);
unsigned int usedA = 0;
int verifyStatus = 0;

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_DSA_HashVerify(&userA, sig, sigSz, ourData,

```

```
CRYPT_SHA_DIGEST_SIZE, &verifyStatus);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
sig	The signature to verify.
sigSz	The length of the signature (octets).
hash	The hash (message digest) that was signed.
hashSz	The length of the hash (octets).
status	Result of signature (1 == valid, 0 == invalid).

Function

```
int CRYPT_ECC_DSA_HashVerify( CRYPT_ECC_CTX* ecc, const unsigned char* sig,
    unsigned int sigSz, unsigned char* hash, unsigned int hashSz, int* status)
```

CRYPT_ECC_Free Function

Cleans up an Elliptic Curve Cryptography (ECC) Context.

File

[crypto.h](#)

C

```
int CRYPT_ECC_Free(CRYPT_ECC_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function cleans up memory allocated for an ECC Context.

Preconditions

The context must have been initialized previously with a call to [CRYPT_ECC_Initialize](#).

Example

```
CRYPT_ECC_CTX userA;
int ret;

ret = CRYPT_ECC_Initialize(&userA);
...
ret = CRYPT_ECC_Free(&userA);
```

Parameters

Parameters	Description
ecc	Pointer to context to clean up.

Function

```
int CRYPT_ECC_Free( CRYPT_ECC_CTX* ecc)
```

CRYPT_ECC_Initialize Function

Initializes the context for Elliptic Curve Cryptography (ECC).

File

[crypto.h](#)

C

```
int CRYPT_ECC_Initialize(CRYPT_ECC_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- -1 - Unable to allocate memory for the keys.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the context used for Elliptic Curve Cryptography (ECC). The context is then passed to calls to perform key building, encryption, decryption, etc.

Preconditions

None.

Example

```
CRYPT_ECC_CTX userA;
int          ret;

ret = CRYPT_ECC_Initialize(&userA);
```

Parameters

Parameters	Description
ecc	Pointer to context to initialize.

Function

```
int CRYPT_ECC_Initialize( CRYPT\_ECC\_CTX\* ecc)
```

CRYPT_ECC_KeySizeGet Function

Returns the key size in octets.

File

[crypto.h](#)

C

```
int CRYPT_ECC_KeySizeGet(CRYPT_ECC_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - The size of the key, in octets.

Description

This function returns the size of the ECC key, in octets.

Preconditions

The ECC context must have been initialized with a call to [CRYPT_ECC_Initialize](#). The key must have been imported or created prior to calling this function.

Example

```
CRYPT_ECC_CTX userA;
int          ret;
byte         sig[100];
unsigned int  sigSz = (unsigned int)sizeof(sig);
```

```

unsigned int   usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_KeySizeGet(&userA);

```

Parameters

Parameters	Description
<code>ecc</code>	Pointer to context which saves state between calls and contains the key.

Function

```
int CRYPT_ECC_KeySizeGet( CRYPT\_ECC\_CTX\* ecc)
```

CRYPT_ECC_PrivateImport Function

Imports private key pair in X9.63 format.

File

[crypto.h](#)

C

```

int CRYPT_ECC_PrivateImport(CRYPT\_ECC\_CTX\*, const unsigned char*, unsigned int, const unsigned char*, unsigned int);

```

Returns

- `BAD_FUNC_ARG` - An invalid pointer was passed to the function.
- `0` - An invalid pointer was not passed to the function.

Description

This function imports a public/private key pair in X9.63 format.

Preconditions

The context must have been initialized with a call to [CRYPT_ECC_Initialize](#).

Example

```

CRYPT_ECC_CTX ecc;

CRYPT_ECC_Initialize(&ecc);
...
CRYPT_ECC_PrivateImport(&ecc, priv_key, sizeof(priv_key), pub_key, sizeof(pub_key));

```

Parameters

Parameters	Description
<code>ecc</code>	Pointer to context which saves state between calls.
<code>priv</code>	Pointer to the private key.
<code>privSz</code>	Size of the private key, in bytes.
<code>pub</code>	Pointer to the public key.
<code>pubSz</code>	Size of the public key, in bytes.

Function

```

int CRYPT_ECC_PrivateImport( CRYPT\_ECC\_CTX\* ecc, const unsigned char* priv,
unsigned int privSz, const unsigned char* pub,
unsigned int pubSz)

```

CRYPT_ECC_PublicExport Function

Exports public ECC key in ANSI X9.63 format.

File

[crypto.h](#)

C

```
int CRYPT_ECC_PublicExport(CRYPT_ECC_CTX*, unsigned char*, unsigned int, unsigned int*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- BUFFER_E - The output buffer was too small to store the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function takes an ECC public key and exports it in ANSI X9.63 format.

Preconditions

The context must be initialized previously with a call to [CRYPT_ECC_Initialize](#). The key must also have been constructed with a call to [CRYPT_ECC_DHE_KeyMake](#). A random number generator must all have been initialized with a call to [CRYPT_RNG_Initialize](#).

Example

```
CRYPT_ECC_CTX userA;  
int          ret;  
byte         sharedA[100];  
unsigned int  aSz = (unsigned int)sizeof(sharedA);  
unsigned int  usedA = 0;  
  
ret = CRYPT_ECC_Initialize(&userA);  
ret = CRYPT_ECC_DHE_KeyMake(&userA, &mcRng, 32);  
ret = CRYPT_ECC_PublicExport(&userA, sharedA, aSz, &usedA);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
out	Buffer in which to store the public key.
outSz	The available size of the buffer, in bytes.
usedSz	Return value indicating how many bytes were used.

Function

```
int CRYPT_ECC_PublicExport( CRYPT_ECC_CTX* ecc, unsigned char* out,  
                           unsigned int outSz, unsigned int* usedSz)
```

CRYPT_ECC_PublicImport Function

Imports public key in ANSI X9.63 format.

File

[crypto.h](#)

C

```
int CRYPT_ECC_PublicImport(CRYPT_ECC_CTX*, const unsigned char*, unsigned int);
```


Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- MEMORY_E - Memory could not be allocated for the key.
- ASN_PARSE_E - There was a parse error while going through the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function imports a public key in ANSI X9.63 format into the ECC context.

Preconditions

The ECC context must have previously been initialized with a call to [CRYPT_ECC_Initialize](#).

Example

```
CRYPT_ECC_CTX userB;
int          ret;
byte        sharedA[100];
unsigned int aSz = (unsigned int)sizeof(sharedA);
unsigned int usedA;

ret = CRYPT_ECC_Initialize(&userB);
...
ret = CRYPT_ECC_PublicImport(&userB, sharedA, usedA);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls.
in	Input buffer the holds the public key.
inSz	Size of the input buffer, in bytes.

Function

```
int CRYPT_ECC_PublicImport( CRYPT\_ECC\_CTX\* ecc, const unsigned char* in,
                           unsigned int inSz)
```

CRYPT_ECC_SignatureSizeGet Function

Returns the signature size in octets.

File

[crypto.h](#)

C

```
int CRYPT_ECC_SignatureSizeGet(CRYPT_ECC_CTX* );
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - The size of the signature.

Description

This function returns the size of the signature in a given context, in octets.

Preconditions

The ECC context must have been initialized with a call to [CRYPT_ECC_Initialize](#). The keys must have been imported or created prior to calling this function.

Example

```
CRYPT_ECC_CTX userA;
```

```
int         ret;
byte        sig[100];
unsigned int sigSz = (unsigned int)sizeof(sig);
unsigned int usedA = 0;

ret = CRYPT_ECC_Initialize(&userA);
...
// Import or generate private key
...
ret = CRYPT_ECC_SignatureSizeGet(&userA);
```

Parameters

Parameters	Description
ecc	Pointer to context which saves state between calls, and contains the signature.

Function

```
int CRYPT_ECC_SignatureSizeGet( CRYPT_ECC_CTX* ecc)
```

g) RSA Encryption/Decryption Functions

CRYPT_RSA_EncryptSizeGet Function

Gets the size of the RSA Key.

File

[crypto.h](#)

C

```
int CRYPT_RSA_EncryptSizeGet(CRYPT_RSA_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - Size of the key.

Description

This function retrieves the size of the RSA Key in use for the context.

Preconditions

The context must be initialized with a call to [CRYPT_RSA_Initialize](#) and the keys decoded either with [CRYPT_RSA_PrivateKeyDecode](#) or [CRYPT_RSA_PublicKeyDecode](#).

Example

```
CRYPT_RSA_CTX mcRsa;
int         ret;
unsigned int keySz = (unsigned int)sizeof(client_key_der_1024);
byte        out1[256];

ret = CRYPT_RSA_Initialize(&mcRsa);

ret = CRYPT_RSA_PrivateKeyDecode(&mcRsa, client_key_der_1024, keySz);

ret = CRYPT_RSA_EncryptSizeGet(&mcRsa);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.

Function

```
int CRYPT_RSA_EncryptSizeGet( CRYPT_RSA_CTX* rsa)
```

CRYPT_RSA_Free Function

Releases the memory used for the key and cleans up the context.

File

[crypto.h](#)

C

```
int CRYPT_RSA_Free(CRYPT_RSA_CTX* );
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function releases the memory used during RSA processing for storing the public/private key.

Preconditions

The context must have been set up previously with a call to [CRYPT_RSA_Initialize](#).

Example

```
CRYPT_RSA_CTX mcRsa;  
int          ret;  
  
ret = CRYPT_RSA_Initialize(&mcRsa);  
ret = CRYPT_RSA_Free(&mcRsa);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.

Function

```
int CRYPT_RSA_Free( CRYPT_RSA_CTX* rsa)
```

CRYPT_RSA_Initialize Function

Initializes the internal structures necessary for RSA processing.

File

[crypto.h](#)

C

```
int CRYPT_RSA_Initialize(CRYPT_RSA_CTX* );
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- -1 - Unable to allocate the memory necessary for the key.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the context used during public-key RSA encryption and decryption.

Preconditions

None.

Example

```
CRYPT_RSA_CTX mcRsa;
int          ret;

ret = CRYPT_RSA_Initialize(&mcRsa);
```

Parameters

Parameters	Description
rsa	Pointer to RSA context which saves state between calls.

Function

int CRYPT_RSA_Initialize([CRYPT_RSA_CTX*](#) rsa)

CRYPT_RSA_PrivateDecrypt Function

Decrypts data using a private key.

File

[crypto.h](#)

C

```
int CRYPT_RSA_PrivateDecrypt(CRYPT_RSA_CTX*, unsigned char*, unsigned int, const unsigned
char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - Size of the actual output.

Description

This function decrypts a data block using a private key.

Preconditions

The context must be initialized using CRYPT_RSA_Initialized and the Private Key Decoded using [CRYPT_RSA_PrivateKeyDecode](#) prior to calling this function.

Example

```
CRYPT_RSA_CTX mcRsa;
int          ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);
byte         out1[256];

ret = CRYPT_RSA_Initialize(&mcRsa);

ret = CRYPT_RSA_PrivateKeyDecode(&mcRsa, client_key_der_1024, keySz);

ret = CRYPT_RSA_PrivateDecrypt(&mcRsa, out2, sizeof(out2), out1,
                              RSA_TEST_SIZE);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
out	Pointer to output buffer to store results.
outSz	Size of output buffer.
in	Pointer to source buffer to be decrypted.

inSz	Size of input buffer.
------	-----------------------

Function

int CRYPT_RSA_PrivateDecrypt([CRYPT_RSA_CTX*](#) rsa, unsigned char* out,
unsigned int outSz, const unsigned char* in, unsigned int inSz)

CRYPT_RSA_PrivateKeyDecode Function

Constructs the Private Key from a DER certificate.

File

[crypto.h](#)

C

```
int CRYPT_RSA_PrivateKeyDecode(CRYPT_RSA_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function builds a private key from a DER-formatted certificate. DER stands for Distinguished Encoding Rules.

Preconditions

The context must have been initialized with a call to [CRYPT_RSA_Initialize](#).

Example

```
CRYPT_RSA_CTX mcRsa;
int          ret;
unsigned int  keySz = (unsigned int)sizeof(client_key_der_1024);

ret = CRYPT_RSA_Initialize(&mcRsa);

ret = CRYPT_RSA_PrivateKeyDecode(&mcRsa, client_key_der_1024, keySz);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
in	Pointer to buffer containing the certificate to process to extract the private key.
inSz	Size of the input data in bytes.

Function

int CRYPT_RSA_PrivateKeyDecode([CRYPT_RSA_CTX*](#), const unsigned char*,
unsigned int)

CRYPT_RSA_PublicEncrypt Function

Encrypts data using a public key.

File

[crypto.h](#)

C

```
int CRYPT_RSA_PublicEncrypt(CRYPT_RSA_CTX*, unsigned char*, unsigned int, const unsigned char*,  
unsigned int, CRYPT_RNG_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- int - Size of the actual output.

Description

This function encrypts a data block using a public key.

Preconditions

The context must be initialized using CRYPT_RSA_Initialized and the Public Key Decoded using [CRYPT_RSA_PublicKeyDecode](#) prior to calling this function. The random number generator must be initialized with a call to [CRYPT_RNG_Initialize](#).

Example

```
CRYPT_RSA_CTX mcRsa;
CRYPT_RNG_CTX mcRng;
int ret;
unsigned int keySz = (unsigned int)sizeof(client_key_der_1024);
byte out1[256];

ret = CRYPT_RSA_Initialize(&mcRsa);
ret = CRYPT_RNG_Initialize(&mcRng);

ret = CRYPT_RSA_PublicKeyDecode(&mcRsa, client_key_der_1024, keySz);

ret = CRYPT_RSA_PublicEncrypt(&mcRsa, out1, sizeof(out1), ourData,
                             RSA_TEST_SIZE, &mcRng);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
out	Pointer to output buffer to store results.
outSz	Size of output buffer.
in	Pointer to source buffer to be encrypted.
inSz	Size of input buffer.
rng	Pointer to Random Number Generator (RNG) context.

Function

```
int CRYPT_RSA_PublicEncrypt( CRYPT_RSA_CTX* rsa, unsigned char* out,
                             unsigned int outSz, const unsigned char* in, unsigned int inSz,
                             CRYPT_RNG_CTX* rng)
```

CRYPT_RSA_PublicKeyDecode Function

Constructs the Public Key from a DER certificate.

File

[crypto.h](#)

C

```
int CRYPT_RSA_PublicKeyDecode(CRYPT_RSA_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function builds a public key from a DER-formatted certificate. DER stands for Distinguished Encoding Rules.

Preconditions

The context must have been initialized with a call to [CRYPT_RSA_Initialize](#).

Example

```
CRYPT_RSA_CTX mcRsa;  
int ret;  
unsigned int keySz = (unsigned int)sizeof(client_key_der_1024);  
  
ret = CRYPT_RSA_Initialize(&mcRsa);  
  
ret = CRYPT_RSA_PublicKeyDecode(&mcRsa, client_key_der_1024, keySz);
```

Parameters

Parameters	Description
rsa	Pointer to context which saves state between calls.
in	Pointer to buffer containing the certificate to process to extract the public key.
inSz	Size of the input data in bytes.

Function

int CRYPT_RSA_PublicKeyDecode([CRYPT_RSA_CTX](#)* rsa, const unsigned char* in,
unsigned int inSz)

h) Triple DES (3DES) Encryption/Decryption Functions

CRYPT_TDES_CBC_Decrypt Function

Decrypts a data block using Triple DES.

File

[crypto.h](#)

C

```
int CRYPT_TDES_CBC_Decrypt(CRYPT_TDES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function decrypts a block of data using a Triple DES algorithm.

Remarks

Input data must have a length a multiple of 8 bytes. Output data will be zero-padded at the end if the original data was not a multiple of 8 bytes long.

Preconditions

The context tdes must be set earlier using [CRYPT_TDES_KeySet](#). The input block must be a multiple of 8 bytes long.

Example

```
CRYPT_TDES_CTX mcDes3;  
int ret;  
byte out1[TDES_SIZE];  
byte out2[TDES_SIZE];
```

```

strncpy((char*)key, "1234567890abcdefghijklmn", 24);
strncpy((char*)iv, "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);

ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);

```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
out	Pointer to output buffer to store the results.
in	Pointer to input buffer for the source of the data.
inSz	Size of the input data buffer.

Function

```
int CRYPT_TDES_CBC_Decrypt( CRYPT_TDES_CTX* tdes, unsigned char* out, const unsigned char* in, unsigned int inSz)
```

CRYPT_TDES_CBC_Encrypt Function

Encrypts a data block using Triple DES.

File

[crypto.h](#)

C

```
int CRYPT_TDES_CBC_Encrypt(CRYPT_TDES_CTX*, unsigned char*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function encrypts a block of data using a Triple DES algorithm.

Remarks

The input data must be padded at the end with zeros to make the length a multiple of 8.

Preconditions

The context tdes must be set earlier using [CRYPT_TDES_KeySet](#). The input block must be a multiple of 8 bytes long.

Example

```

CRYPT_TDES_CTX mcDes3;
int ret;
byte out1[TDES_SIZE];
byte out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmn", 24);
strncpy((char*)iv, "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);

```



```
ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
out	Pointer to output buffer to store the results.
in	Pointer to input buffer for the source of the data.
inSz	Size of the input data buffer.

Function

```
int CRYPT_TDES_CBC_Encrypt( CRYPT_TDES_CTX* tdes, unsigned char* out, const unsigned char* in, unsigned int inSz)
```

CRYPT_TDES_IvSet Function

Sets the Initialization Vector (IV) for a Triple DES operation.

File

[crypto.h](#)

C

```
int CRYPT_TDES_IvSet( CRYPT_TDES_CTX*, const unsigned char* );
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function changes the IV of a TDES context, but leaves the Key alone.

Remarks

The IV must be 8 bytes long.

Preconditions

None.

Example

```
CRYPT_TDES_CTX mcDes3;
int           ret;
byte          out1[TDES_SIZE];
byte          out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuvwxyz", 24);
strncpy((char*)iv, "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_IvSet(&mcDes3, iv);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);

ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
iv	Pointer to buffer holding the initialization vector. Must be 8 bytes in size.

Function

int CRYPT_TDES_IvSet([CRYPT_TDES_CTX](#)* tdes, const unsigned char* iv)

CRYPT_TDES_KeySet Function

Initialization of Triple DES context.

File

[crypto.h](#)

C

```
int CRYPT_TDES_KeySet(CRYPT_TDES_CTX*, const unsigned char*, const unsigned char*, int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function sets the key and initialization vector (IV) for a set of Triple-DES operations.

Remarks

The input data must be a multiple of 8 bytes, and must be padded at the end with zeros to meet the length.

Preconditions

None.

Example

```
CRYPT_TDES_CTX mcDes3;
int           ret;
byte          out1[TDES_SIZE];
byte          out2[TDES_SIZE];

strncpy((char*)key, "1234567890abcdefghijklmnopqrstuvwxyz", 24);
strncpy((char*)iv,  "12345678", 8);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_ENCRYPTION);

ret = CRYPT_TDES_CBC_Encrypt(&mcDes3, out1, ourData, TDES_SIZE);

ret = CRYPT_TDES_KeySet(&mcDes3, key, iv, CRYPT_TDES_DECRYPTION);

ret = CRYPT_TDES_CBC_Decrypt(&mcDes3, out2, out1, TDES_TEST_SIZE);
```

Parameters

Parameters	Description
tdes	Pointer to context which saves state between calls.
key	Pointer to buffer holding the key. Must be 24 bytes in size.
iv	Pointer to buffer holding the initialization vector. Must be 8 bytes in size.
dir	Indicates whether encryption or decryption will be done. Can be set to: <ul style="list-style-type: none">• CRYPT_TDES_ENCRYPTION - For Encryption operations• CRYPT_TDES_DECRYPTION - Fro Decryption operations

Function

int CRYPT_TDES_KeySet([CRYPT_TDES_CTX](#)* tdes, const unsigned char* key, const unsigned char* iv, int dir)

i) HMAC Hash Functions

CRYPT_HMAC_DataAdd Function

Adds data to the HMAC calculation.

File

[crypto.h](#)

C

```
int CRYPT_HMAC_DataAdd(CRYPT_HMAC_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function adds data to the HMAC so that multiple blocks of data can be processed.

Remarks

None.

Preconditions

The [CRYPT_HMAC_CTX](#) context must be initialized using the [CRYPT_HMAC_SetKey](#) function prior to any call to this function.

Example

```
CRYPT_HMAC_CTX mcHmac;  
byte          mcDigest[CRYPT_SHA512_DIGEST_SIZE];  
  
CRYPT_HMAC_SetKey(&mcHmac, CRYPT_HMAC_SHA, key, 4);  
  
CRYPT_HMAC_DataAdd(&mcHmac, ourData, OUR_DATA_SIZE);  
  
CRYPT_HMAC_Finalize(&mcHmac, mcDigest);
```

Parameters

Parameters	Description
hmac	Pointer to context that saves state between calls.
sz	Size of the input data in bytes.

Function

```
int CRYPT_HMAC_DataAdd( CRYPT_HMAC_CTX*, const unsigned char*, unsigned int)
```

CRYPT_HMAC_Finalize Function

Completes the HMAC calculation and get the results.

File

[crypto.h](#)

C

```
int CRYPT_HMAC_Finalize(CRYPT_HMAC_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function completes the HMAC calculations. The results are placed in the location pointed to by the digest parameter.

Remarks

The area pointed to by the digest parameter must be large enough to hold the results.

Preconditions

The [CRYPT_HMAC_CTX](#) context must be initialized using the [CRYPT_HMAC_SetKey](#) function prior to any call to this function.

Example

```
CRYPT_HMAC_CTX mcHmac;  
byte          mcDigest[CRYPT_SHA512_DIGEST_SIZE];  
  
CRYPT_HMAC_SetKey(&mcHmac, CRYPT_HMAC_SHA, key, 4);  
  
CRYPT_HMAC_DataAdd(&mcHmac, ourData, OUR_DATA_SIZE);  
  
CRYPT_HMAC_Finalize(&mcHmac, mcDigest);
```

Parameters

Parameters	Description
hmac	Pointer to context which saves state between calls.
digest	Pointer to place to put the final HMAC digest results.

Function

int CRYPT_HMAC_Finalize([CRYPT_HMAC_CTX*](#) hmac, unsigned char* digest)

CRYPT_HMAC_SetKey Function

Initializes the HMAC context and set the key for the hash.

File

[crypto.h](#)

C

```
int CRYPT_HMAC_SetKey(CRYPT_HMAC_CTX*, int, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the HMAC context and set the key for the hash.

Remarks

None.

Preconditions

None.

Example

```
CRYPT_HMAC_CTX mcHmac;  
byte          mcDigest[CRYPT_SHA512_DIGEST_SIZE];
```

```
CRYPT_HMAC_SetKey(&mcHmac, CRYPT_HMAC_SHA, key, 4);

CRYPT_HMAC_DataAdd(&mcHmac, ourData, OUR_DATA_SIZE);

CRYPT_HMAC_Finalize(&mcHmac, mcDigest);
```

Parameters

Parameters	Description
hmac	Pointer to context which saves state between calls.
type	Type of SHA operation to use with HMAC. Must be one of the
following	<ul style="list-style-type: none">• CRYPT_HMAC_SHA• CRYPT_HMAC_SHA256• CRYPT_HMAC_SHA384• CRYPT_HMAC_SHA512
key	Secret key used for the hash.
sz	Size of the input data in bytes.

Function

```
int CRYPT_HMAC_SetKey( CRYPT_HMAC_CTX* hmac, int type, const unsigned char* key, unsigned int sz)
```

j) SHA Hash functions

CRYPT_SHA_DataAdd Function

Updates the hash with the data provided.

File

[crypto.h](#)

C

```
int CRYPT_SHA_DataAdd(CRYPT_SHA_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA hash, nothing must modify the context holding variable between calls to CRYPT_SHA_DataAdd.

Preconditions

The SHA context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA_CTX sha;
uint8_t buffer[1024];
uint8_t shaSum[SHA_DIGEST_SIZE];

CRYPT_SHA_Initialize(&sha);
```

```
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));  
CRYPT_SHA_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

```
int CRYPT_SHA_DataAdd( CRYPT\_SHA\_CTX* sha, const unsigned char* input, unsigned int sz)
```

CRYPT_SHA_Finalize Function

Finalizes the hash and puts the result into digest.

File

[crypto.h](#)

C

```
int CRYPT_SHA_Finalize(CRYPT_SHA_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA hash, nothing must modify the context holding variable between calls to [CRYPT_SHA_DataAdd](#) and [CRYPT_SHA_Finalize](#).

Preconditions

The SHA context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA_CTX sha;  
uint8_t buffer[1024];  
uint8_t shaSum[SHA_DIGEST_SIZE];  
  
CRYPT_SHA_Initialize(&sha);  
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));  
CRYPT_SHA_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

```
int CRYPT_SHA_Finalize( CRYPT\_SHA\_CTX* sha, unsigned char* digest)
```

CRYPT_SHA_Initialize Function

Initializes the internal structures necessary for SHA hash calculations.

File

[crypto.h](#)

C

```
int CRYPT_SHA_Initialize(CRYPT_SHA_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for SHA hash calculations.

Remarks

All SHA hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA_CTX sha;
uint8_t shaSum[SHA_DIGEST_SIZE];

CRYPT_SHA_Initialize(&sha);
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.

Function

```
int CRYPT_SHA_Initialize( CRYPT\_SHA\_CTX\* sha)
```

CRYPT_SHA_DataSizeSet Function

For PIC32MZ hardware encryption, sets the size of the input data.

File

[crypto.h](#)

C

```
int CRYPT_SHA_DataSizeSet(CRYPT_SHA_CTX*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

The PIC32MZ hardware encryption module needs to know the size of the data before it starts processing. This function sets that value.

Remarks

All SHA hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA_CTX sha;
uint8_t buffer[1024];
uint8_t shasum[SHA_DIGEST_SIZE];

CRYPT_SHA_Initialize(&sha);
CRYPT_SHADataSizeSet(&sha, sizeof(buffer));
CRYPT_SHA_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA_Finalize(&sha, shasum);
```

Parameters

Parameters	Description
sha	Pointer to CRYPT_SHA_CTX structure which holds the hash values.
msgSize	Size of the data (in bytes) that will be processed.

Function

```
int CRYPT_SHA_DataSizeSet( CRYPT_SHA_CTX* sha, unsigned int msgSize)
```

k) SHA256 Hash Functions

CRYPT_SHA256_DataAdd Function

Updates the hash with the data provided.

File

[crypto.h](#)

C

```
int CRYPT_SHA256_DataAdd(CRYPT_SHA256_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha256 or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA256 hash, nothing must modify the context holding variable between calls to `CRYPT_SHA256_DataAdd`.

Preconditions

The SHA256 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA256_CTX sha256;
uint8_t buffer[1024];
uint8_t shaSum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha256);
CRYPT_SHA256_DataAdd(&sha256, buffer, sizeof(buffer));
```



```
CRYPT_SHA256_Finalize(&sha256, shaSum);
```

Parameters

Parameters	Description
sha256	Pointer to CRYPT_SHA256_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

```
int CRYPT_SHA256_DataAdd( CRYPT\_SHA256\_CTX* sha256, const unsigned char* input, unsigned int sz)
```

CRYPT_SHA256_Finalize Function

Finalizes the hash and puts the result into digest.

File

[crypto.h](#)

C

```
int CRYPT_SHA256_Finalize(CRYPT_SHA256_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA256 hash, nothing must modify the context holding variable between calls to [CRYPT_SHA256_DataAdd](#) and [CRYPT_SHA256_Finalize](#).

Preconditions

The SHA256 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA256_CTX sha256;
uint8_t buffer[1024];
uint8_t shaSum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha256);
CRYPT_SHA256_DataAdd(&sha256, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha256, shaSum);
```

Parameters

Parameters	Description
sha256	Pointer to CRYPT_SHA256_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

```
int CRYPT_SHA256_Finalize( CRYPT\_SHA256\_CTX* sha256, unsigned char* digest)
```

CRYPT_SHA256_Initialize Function

Initializes the internal structures necessary for SHA256 hash calculations.

File

[crypto.h](#)

C

```
int CRYPT_SHA256_Initialize(CRYPT_SHA256_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for SHA256 hash calculations.

Remarks

All SHA hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA256_CTX sha;
uint8_t shaSum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha);
CRYPT_SHA256_DataAdd(&sha, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha, shaSum);
```

Parameters

Parameters	Description
sha256	Pointer to context which saves state between calls.

Function

```
int CRYPT_SHA256_Initialize( CRYPT_SHA256_CTX* sha256)
```

CRYPT_SHA256_DataSizeSet Function

For PIC32MZ hardware encryption, sets the size of the input data.

File

[crypto.h](#)

C

```
int CRYPT_SHA256_DataSizeSet(CRYPT_SHA256_CTX*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

The PIC32MZ hardware encryption module needs to know the size of the data before it starts processing. This function sets that value.

Remarks

All SHA256 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA256_CTX sha256;
uint8_t buffer[1024];
uint8_t sha256sum[SHA256_DIGEST_SIZE];

CRYPT_SHA256_Initialize(&sha256);
CRYPT_SHA256DataSizeSet(&sha256, sizeof(buffer));
CRYPT_SHA256_DataAdd(&sha256, buffer, sizeof(buffer));
CRYPT_SHA256_Finalize(&sha256, sha256sum);
```

Parameters

Parameters	Description
sha256	Pointer to CRYPT_SHA256_CTX structure which holds the hash values.
msgSize	Size of the data (in bytes) that will be processed.

Function

```
int CRYPT_SHA256_DataSizeSet( CRYPT_SHA256_CTX* sha256, unsigned int msgSize)
```

I) SHA384 Hash Functions

CRYPT_SHA384_DataAdd Function

Updates the hash with the data provided.

File

[crypto.h](#)

C

```
int CRYPT_SHA384_DataAdd(CRYPT_SHA384_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha384 or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA384 hash, nothing must modify the context holding variable between calls to `CRYPT_SHA384_DataAdd`.

Preconditions

The SHA384 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA384_CTX sha384;
uint8_t buffer[1024];
uint8_t shaSum[SHA384_DIGEST_SIZE];

CRYPT_SHA384_Initialize(&sha384);
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));
```

```
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

```
int CRYPT_SHA384_DataAdd( CRYPT\_SHA384\_CTX* sha384, const unsigned char* input, unsigned int sz)
```

CRYPT_SHA384_Finalize Function

Finalizes the hash and puts the result into digest.

File

[crypto.h](#)

C

```
int CRYPT_SHA384_Finalize(CRYPT_SHA384_CTX*, unsigned char*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha384 or digest.
- 0 - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA384 hash, nothing must modify the context holding variable between calls to [CRYPT_SHA384_DataAdd](#) and [CRYPT_SHA384_Finalize](#).

Preconditions

The SHA384 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA384_CTX sha384;
uint8_t buffer[1024];
uint8_t shaSum[SHA384_DIGEST_SIZE];

CRYPT_SHA384_Initialize(&sha384);
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

```
int CRYPT_SHA384_Finalize( CRYPT\_SHA384\_CTX* sha384, unsigned char* digest)
```

CRYPT_SHA384_Initialize Function

Initializes the internal structures necessary for SHA384 hash calculations.

File

[crypto.h](#)

C

```
int CRYPT_SHA384_Initialize(CRYPT_SHA384_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function
- 0 - An invalid pointer was not passed to the function

Description

This function initializes the internal structures necessary for SHA384 hash calculations.

Remarks

All SHA384 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA384_CTX sha384;
uint8_t shaSum[SHA384_DIGEST_SIZE];

CRYPT_SHA384_Initialize(&sha384);
CRYPT_SHA384_DataAdd(&sha384, buffer, sizeof(buffer));
CRYPT_SHA384_Finalize(&sha384, shaSum);
```

Parameters

Parameters	Description
sha384	Pointer to CRYPT_SHA384_CTX structure which holds the hash values.

Function

```
int CRYPT_SHA384_Initialize( CRYPT\_SHA384\_CTX\* sha384)
```

m) SHA512 Hash Functions

CRYPT_SHA512_DataAdd Function

Updates the hash with the data provided.

File

[crypto.h](#)

C

```
int CRYPT_SHA512_DataAdd(CRYPT_SHA512_CTX*, const unsigned char*, unsigned int);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function, either in sha512 or input.
- 0 - An invalid pointer was not passed to the function.

Description

This function updates the hash with the data provided.

Remarks

In order to preserve the validity of the SHA512 hash, nothing must modify the context holding variable between calls to `CRYPT_SHA512_DataAdd`.

Preconditions

The SHA512 context must be initialized prior to the first call of this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA512_CTX sha512;
uint8_t buffer[1024];
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

Parameters

Parameters	Description
sha512	Pointer to <code>CRYPT_SHA512_CTX</code> structure which holds the hash values.
sz	Size of the data (in bytes) of the data to use to update the hash.

Function

```
int CRYPT_SHA512_DataAdd( CRYPT_SHA512_CTX* sha512, const unsigned char* input, unsigned int sz)
```

CRYPT_SHA512_Finalize Function

Finalizes the hash and puts the result into digest.

File

[crypto.h](#)

C

```
int CRYPT_SHA512_Finalize(CRYPT_SHA512_CTX*, unsigned char*);
```

Returns

- `BAD_FUNC_ARG` - An invalid pointer was passed to the function, either in sha512 or digest.
- `0` - An invalid pointer was not passed to the function.

Description

This function finalizes the hash and puts the result into digest.

Remarks

In order to preserve the validity of the SHA512 hash, nothing must modify the context holding variable between calls to [CRYPT_SHA512_DataAdd](#) and `CRYPT_SHA512_Finalize`.

Preconditions

The SHA512 context must be initialized prior to calling this function. The context must not be modified by code outside of this function.

Example

```
CRYPT_SHA512_CTX sha512;
uint8_t buffer[1024];
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
```

```
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

Parameters

Parameters	Description
sha512	Pointer to CRYPT_SHA512_CTX structure which holds the hash values.
digest	Pointer to byte array to store hash result.

Function

```
int CRYPT_SHA512_Finalize( CRYPT\_SHA512\_CTX* sha512, unsigned char* digest)
```

CRYPT_SHA512_Initialize Function

Initializes the internal structures necessary for SHA512 hash calculations.

File

[crypto.h](#)

C

```
int CRYPT_SHA512_Initialize(CRYPT_SHA512_CTX*);
```

Returns

- BAD_FUNC_ARG - An invalid pointer was passed to the function.
- 0 - An invalid pointer was not passed to the function.

Description

This function initializes the internal structures necessary for SHA512 hash calculations.

Remarks

All SHA512 hashes have to start at a particular value before adding new data to it. This function sets the necessary values for the structure.

Preconditions

None.

Example

```
CRYPT_SHA512_CTX sha512;
uint8_t sha512Sum[SHA512_DIGEST_SIZE];

CRYPT_SHA512_Initialize(&sha512);
CRYPT_SHA512_DataAdd(&sha512, buffer, sizeof(buffer));
CRYPT_SHA512_Finalize(&sha512, sha512Sum);
```

Parameters

Parameters	Description
sha512	Pointer to CRYPT_SHA512_CTX structure which holds the hash values.

Function

```
int CRYPT_SHA512_Initialize( CRYPT\_SHA512\_CTX* sha512)
```

n) Data Types and Constants

CRYPT_AES_CTX Structure

File

[crypto.h](#)

C

```
struct CRYPT_AES_CTX {  
    int holder[90];  
};
```

Members

Members	Description
int holder[90];	This structure should be large enough to hold the internal representation, the size is checked during initialization

Description

AES

CRYPT_ECC_CTX Structure

File

[crypto.h](#)

C

```
struct CRYPT_ECC_CTX {  
    void* holder;  
};
```

Description

ECC

CRYPT_HMAC_CTX Structure

File

[crypto.h](#)

C

```
struct CRYPT_HMAC_CTX {  
    long long holder[80];  
};
```

Members

Members	Description
long long holder[80];	This structure should be large enough to hold the internal representation, the size is checked during initialization

Description

HMAC

CRYPT_MD5_CTX Structure

File

[crypto.h](#)

C

```
struct CRYPT_MD5_CTX {  
    int holder[110];  
};
```

Members

Members	Description
int holder[110];	This structure should be large enough to hold the internal representation, the size is checked during initialization

Description

MD5

CRYPT_RNG_CTX Structure

File

[crypto.h](#)

C

```
struct CRYPT_RNG_CTX {  
    int holder[66];  
};
```

Members

Members	Description
int holder[66];	This structure should be large enough to hold the internal representation, the size is checked during initialization

Description

RNG

CRYPT_RSA_CTX Structure

File

[crypto.h](#)

C

```
struct CRYPT_RSA_CTX {  
    void* holder;  
};
```

Description

RSA

CRYPT_SHA_CTX Structure

File

crypto.h

C

```
struct CRYPT_SHA_CTX {  
};
```


Description

SHA

CRYPT_SHA_CTX Members

The following tables list the members exposed by CRYPT_SHA_CTX.


Public Methods

	Name	Description
	__attribute__	This structure should be large enough to hold the internal representation, the size is checked during initialization

CRYPT_SHA_CTX Methods

The methods of the CRYPT_SHA_CTX class are listed here.

Public Methods

	Name	Description
	__attribute__	This structure should be large enough to hold the internal representation, the size is checked during initialization

CRYPT_SHA_CTX::__attribute__ Method

C

```
int holder __attribute__((aligned (128)));
```

Description

This structure should be large enough to hold the internal representation, the size is checked during initialization

CRYPT_SHA256_CTX Structure

File

crypto.h

C

```
struct CRYPT_SHA256_CTX {  
};
```


Description

SHA-256

CRYPT_SHA256_CTX Members

The following tables list the members exposed by CRYPT_SHA256_CTX.


Public Methods

	Name	Description
	__attribute__	This structure should be large enough to hold the internal representation, the size is checked during initialization

CRYPT_SHA256_CTX Methods

The methods of the CRYPT_SHA256_CTX class are listed here.

Public Methods

	Name	Description
	__attribute__	This structure should be large enough to hold the internal representation, the size is checked during initialization

CRYPT_SHA256_CTX::__attribute__ Method

C

```
int holder __attribute__((aligned (128)));
```

Description

This structure should be large enough to hold the internal representation, the size is checked during initialization

CRYPT_SHA384_CTX Structure

File

[crypto.h](#)

C

```
struct CRYPT_SHA384_CTX {
    long long holder[32];
};
```

Members

Members	Description
long long holder[32];	This structure should be large enough to hold the internal representation, the size is checked during initialization

Description

SHA-384

CRYPT_SHA512_CTX Structure

File

[crypto.h](#)

C

```
struct CRYPT_SHA512_CTX {  
    long long holder[36];  
};
```

Members

Members	Description
long long holder[36];	This structure should be large enough to hold the internal representation, the size is checked during initialization

Description

SHA-512

CRYPT_TDES_CTX Structure

File

[crypto.h](#)

C

```
struct CRYPT_TDES_CTX {  
    int holder[104];  
};
```

Members

Members	Description
int holder[104];	This structure should be large enough to hold the internal representation, the size is checked during initialization

Description

TDES

MC_CRYPT_API_H Macro

File

[crypto.h](#)

C

```
#define MC_CRYPT_API_H
```

Description

Defines Microchip CRYPTO API layer

Files

Files

Name	Description
crypto.h	Crypto Framework Library header for cryptographic functions.

Description















This section lists the source and header files used by the Crypto Library.

crypto.h

Crypto Framework Library header for cryptographic functions.

Functions












	Name	Description
⇒	CRYPT_AES_CBC_Decrypt	Performs AES decryption using Cipher-Block-Chaining (CBC).
⇒	CRYPT_AES_CBC_Encrypt	Performs AES encryption using Cipher-Block-Chaining (CBC).
⇒	CRYPT_AES_CTR_Encrypt	Performs AES encryption using Counter (CTR).
⇒	CRYPT_AES_DIRECT_Decrypt	Directs decryption of one block of data.
⇒	CRYPT_AES_DIRECT_Encrypt	Directs encryption of one block of data.
⇒	CRYPT_AES_IvSet	Sets the initialization vector (IV) for AES processing.
⇒	CRYPT_AES_KeySet	Sets the key and initialization vector (IV) for AES processing.
⇒	CRYPT_ECC_DHE_KeyMake	Creates a new ECC key.
⇒	CRYPT_ECC_DHE_SharedSecretMake	Creates an ECC shared secret between two keys.
⇒	CRYPT_ECC_DSA_HashSign	Signs a message digest.
⇒	CRYPT_ECC_DSA_HashVerify	Verifies an ECC signature.
⇒	CRYPT_ECC_Free	Cleans up an Elliptic Curve Cryptography (ECC) Context.
⇒	CRYPT_ECC_Initialize	Initializes the context for Elliptic Curve Cryptography (ECC).
⇒	CRYPT_ECC_KeySizeGet	Returns the key size in octets.
⇒	CRYPT_ECC_PrivateImport	Imports private key pair in X9.63 format.
⇒	CRYPT_ECC_PublicExport	Exports public ECC key in ANSI X9.63 format.
⇒	CRYPT_ECC_PublicImport	Imports public key in ANSI X9.63 format.
⇒	CRYPT_ECC_SignatureSizeGet	Returns the signature size in octets.
⇒	CRYPT_ERROR_StringGet	Reports the nature of an error.
⇒	CRYPT_HMAC_DataAdd	Adds data to the HMAC calculation.
⇒	CRYPT_HMAC_Finalize	Completes the HMAC calculation and get the results.
⇒	CRYPT_HMAC_SetKey	Initializes the HMAC context and set the key for the hash.
⇒	CRYPT_HUFFMAN_Compress	Compresses a block of data.
⇒	CRYPT_HUFFMAN_DeCompress	Decompresses a block of data.
⇒	CRYPT_MD5_DataAdd	Updates the hash with the data provided.
⇒	CRYPT_MD5_DataSizeSet	This function sets the size of the input data for use with hardware accelerated encryption.
⇒	CRYPT_MD5_Finalize	Finalizes the hash and puts the result into digest.
⇒	CRYPT_MD5_Initialize	Initializes the internal structures necessary for MD5 hash calculations.
⇒	CRYPT_RNG_BlockGenerate	Create several random numbers.
⇒	CRYPT_RNG_Get	Gets one random number.
⇒	CRYPT_RNG_Initialize	Initializes the random number generator.
⇒	CRYPT_RSA_EncryptSizeGet	Gets the size of the RSA Key.
⇒	CRYPT_RSA_Free	Releases the memory used for the key and cleans up the context.
⇒	CRYPT_RSA_Initialize	Initializes the internal structures necessary for RSA processing.
⇒	CRYPT_RSA_PrivateDecrypt	Decrypts data using a private key.
⇒	CRYPT_RSA_PrivateKeyDecode	Constructs the Private Key from a DER certificate.
⇒	CRYPT_RSA_PublicEncrypt	Encrypts data using a public key.
⇒	CRYPT_RSA_PublicKeyDecode	Constructs the Public Key from a DER certificate.
⇒	CRYPT_SHA_DataAdd	Updates the hash with the data provided.
⇒	CRYPT_SHA_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.
⇒	CRYPT_SHA_Finalize	Finalizes the hash and puts the result into digest.
⇒	CRYPT_SHA_Initialize	Initializes the internal structures necessary for SHA hash calculations.

	CRYPT_SHA256_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA256_DataSizeSet	For PIC32MZ hardware encryption, sets the size of the input data.
	CRYPT_SHA256_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA256_Initialize	Initializes the internal structures necessary for SHA256 hash calculations.
	CRYPT_SHA384_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA384_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA384_Initialize	Initializes the internal structures necessary for SHA384 hash calculations.
	CRYPT_SHA512_DataAdd	Updates the hash with the data provided.
	CRYPT_SHA512_Finalize	Finalizes the hash and puts the result into digest.
	CRYPT_SHA512_Initialize	Initializes the internal structures necessary for SHA512 hash calculations.
	CRYPT_TDES_CBC_Decrypt	Decrypts a data block using Triple DES.
	CRYPT_TDES_CBC_Encrypt	Encrypts a data block using Triple DES.
	CRYPT_TDES_IvSet	Sets the Initialization Vector (IV) for a Triple DES operation.
	CRYPT_TDES_KeySet	Initialization of Triple DES context.

Macros

	Name	Description
	MC_CRYPT_API_H	Defines Microchip CRYPTO API layer

Structures

	Name	Description
	CRYPT_AES_CTX	AES
	CRYPT_ECC_CTX	ECC
	CRYPT_HMAC_CTX	HMAC
	CRYPT_MD5_CTX	MD5
	CRYPT_RNG_CTX	RNG
	CRYPT_RSA_CTX	RSA
	CRYPT_SHA_CTX	SHA
	CRYPT_SHA256_CTX	SHA-256
	CRYPT_SHA384_CTX	SHA-384
	CRYPT_SHA512_CTX	SHA-512
	CRYPT_TDES_CTX	TDES

Description

Crypto Framework Library Header

This header file contains function prototypes and definitions of the data types and constants that make up the Cryptographic Framework Library for PIC32 families of Microchip microcontrollers.

File Name

crypto.h

Company

Microchip Technology Inc.

Index

A

Abstraction Model 8
 Crypto Library 8
 Applications 3

B

Building the Application 3, 5
 Building the Library 9
 Crypto Library 9

C

Configuring the Hardware 3, 6
 Configuring the Library 9
 Crypto Library 9
 CRYPT_AES_CBC_Decrypt function 20
 CRYPT_AES_CBC_Encrypt function 21
 CRYPT_AES_CTR_Encrypt function 21
 CRYPT_AES_CTX structure 56
 CRYPT_AES_DIRECT_Decrypt function 22
 CRYPT_AES_DIRECT_Encrypt function 23
 CRYPT_AES_IvSet function 24
 CRYPT_AES_KeySet function 25
 CRYPT_ECC_CTX structure 56
 CRYPT_ECC_DHE_KeyMake function 25
 CRYPT_ECC_DHE_SharedSecretMake function 26
 CRYPT_ECC_DSA_HashSign function 27
 CRYPT_ECC_DSA_HashVerify function 28
 CRYPT_ECC_Free function 29
 CRYPT_ECC_Initialize function 29
 CRYPT_ECC_KeySizeGet function 30
 CRYPT_ECC_PrivateImport function 31
 CRYPT_ECC_PublicExport function 32
 CRYPT_ECC_PublicImport function 32
 CRYPT_ECC_SignatureSizeGet function 33
 CRYPT_ERROR_StringGet function 12
 CRYPT_HMAC_CTX structure 56
 CRYPT_HMAC_DataAdd function 43
 CRYPT_HMAC_Finalize function 43
 CRYPT_HMAC_SetKey function 44
 CRYPT_HUFFMAN_Compress function 13
 CRYPT_HUFFMAN_DeCompress function 14
 CRYPT_MD5_CTX structure 57
 CRYPT_MD5_DataAdd function 14
 CRYPT_MD5_DataSizeSet function 17
 CRYPT_MD5_Finalize function 15
 CRYPT_MD5_Initialize function 16
 CRYPT_RNG_BlockGenerate function 18
 CRYPT_RNG_CTX structure 57
 CRYPT_RNG_Get function 19
 CRYPT_RNG_Initialize function 18
 CRYPT_RSA_CTX structure 57
 CRYPT_RSA_EncryptSizeGet function 34
 CRYPT_RSA_Free function 35
 CRYPT_RSA_Initialize function 35
 CRYPT_RSA_PrivateDecrypt function 36
 CRYPT_RSA_PrivateKeyDecode function 37

CRYPT_RSA_PublicEncrypt function 37
 CRYPT_RSA_PublicKeyDecode function 38
 CRYPT_SHA_CTX structure 58
 about CRYPT_SHA_CTX structure 58
 CRYPT_SHA_CTX members 58
 CRYPT_SHA_CTX methods 58
 CRYPT_SHA_CTX::__attribute__ method 58
 CRYPT_SHA_DataAdd function 45
 CRYPT_SHA_DataSizeSet function 47
 CRYPT_SHA_Finalize function 46
 CRYPT_SHA_Initialize function 46
 CRYPT_SHA256_CTX structure 58
 about CRYPT_SHA256_CTX structure 58
 CRYPT_SHA256_CTX members 59
 CRYPT_SHA256_CTX methods 59
 CRYPT_SHA256_CTX::__attribute__ method 59
 CRYPT_SHA256_DataAdd function 48
 CRYPT_SHA256_DataSizeSet function 50
 CRYPT_SHA256_Finalize function 49
 CRYPT_SHA256_Initialize function 49
 CRYPT_SHA384_CTX structure 59
 CRYPT_SHA384_DataAdd function 51
 CRYPT_SHA384_Finalize function 52
 CRYPT_SHA384_Initialize function 52
 CRYPT_SHA512_CTX structure 59
 CRYPT_SHA512_DataAdd function 53
 CRYPT_SHA512_Finalize function 54
 CRYPT_SHA512_Initialize function 55
 CRYPT_TDES_CBC_Decrypt function 39
 CRYPT_TDES_CBC_Encrypt function 40
 CRYPT_TDES_CTX structure 60
 CRYPT_TDES_IvSet function 41
 CRYPT_TDES_KeySet function 42
 Crypto Library Help 8
 crypto.h 61

E

Encrypt-Decrypt Demonstration Application 5

F

Files 60
 Crypto Library 60

I

Introduction 8

L

Large Hash Demonstration Application 3
 Library Interface 10
 Crypto Library 10
 Library Overview 9
 Crypto Library 9

M

MC_CRYPT_API_H macro 60

R

Release Information for Crypto 2
 Release Notes 2
 Running the Demonstration 4

Running the Demonstration 6

U

Using the Library 8

 Crypto Library 8