Message Digest Algorithm 5 User Guide

Version 1.50 BETA

For use with Message Digest Algorithm 5 (MD5) module versions 1.10 and above

Date: 22-Feb-2018 10:32

Table of Contents

1 System Overview

This chapter contains the fundamental information for this module.

The component sections are as follows:

- Introduction – describes the main elements of the module.
- Feature Check – summarizes the main features of the module as bullet points.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

1.1 Introduction

This guide is for those who want to generate hash codes for data by using the MD4 or MD5 Message Digest Algorithms. This module implements the MD4 and MD5 hash algorithms. It supports MD4, MD5, HMAC-MD5 (Hash Message Authentication Code), and HMAC-MD5-96.

Note: The MD4 and MD5 methods, which produce a 16 byte hash value, have been superseded by the SHA hash algorithms.

Overview

MD4 and MD5 are hashing algorithms, algorithms that calculate a special number (a hash value) from the data. This value can be checked to validate that the data has not been modified in transit. The variants of this are as follows:

- MD4 - computes a 16 byte hash (this is rarely used but it is used in MS-CHAP, the Microsoft version of the Challenge Handshake Authentication Protocol, CHAP).
- MD5 - computes a 16 byte hash. An OID is specified for MD5 but not for MD4 and HMAC-MD5.
- HMAC-MD5 (HMAC is Hash Message Authentication Code) - uses hashing of data with the addition of a key. This is used in IPsec.
  HMAC-MD5 computes a 16 byte hash and is more secure than MD5 because MD5 strings are short enough to be broken using brute force. HMAC has 64 bytes of padding plus a challenge word.
  To compute HMAC over the data `text' we perform: `H(K XOR opad, H(K XOR ipad, text))` where `H` is a hashing function. (See RFC 2104 page 3.)
- HMAC-MD5-96 - uses as its authentication data the first or leftmost 96 bits of the 128-bit value that HMAC-MD5 produces. This procedure is known as truncation.

HCC's Implementation

For MD4 and MD5 the EEM's **enc_driver_hash()** function is used. For the HMAC variants the EEM's **enc_driver_encrypt()** call is used.

The implementation of all variants is stateful, meaning that you can call it repeatedly to calculate a hash over a large block of data. If the output buffer is set to NULL, this means that more data is to come - if the output buffer *outbuf* is present, the final result is returned and the next call starts.

MD5 challenge requests

MD5 is used in MD5 challenge requests for simple password exchange. This process is as follows:

1. The server sends a challenge string (typically randomly generated) of at least 16 bytes.
2. The client calculates a hash over the challenge and sends the result (plus the password) to the server.
3. The server compares this with its stored password and hash combination.

MD5 and MD4 Usage

MD5 computes a 16 byte hash. An OID is specified for MD5 but not for MD4 and HMAC-MD5.

The EEM function **enc_driver_hash()** is used to calculate the hash value of the given data.

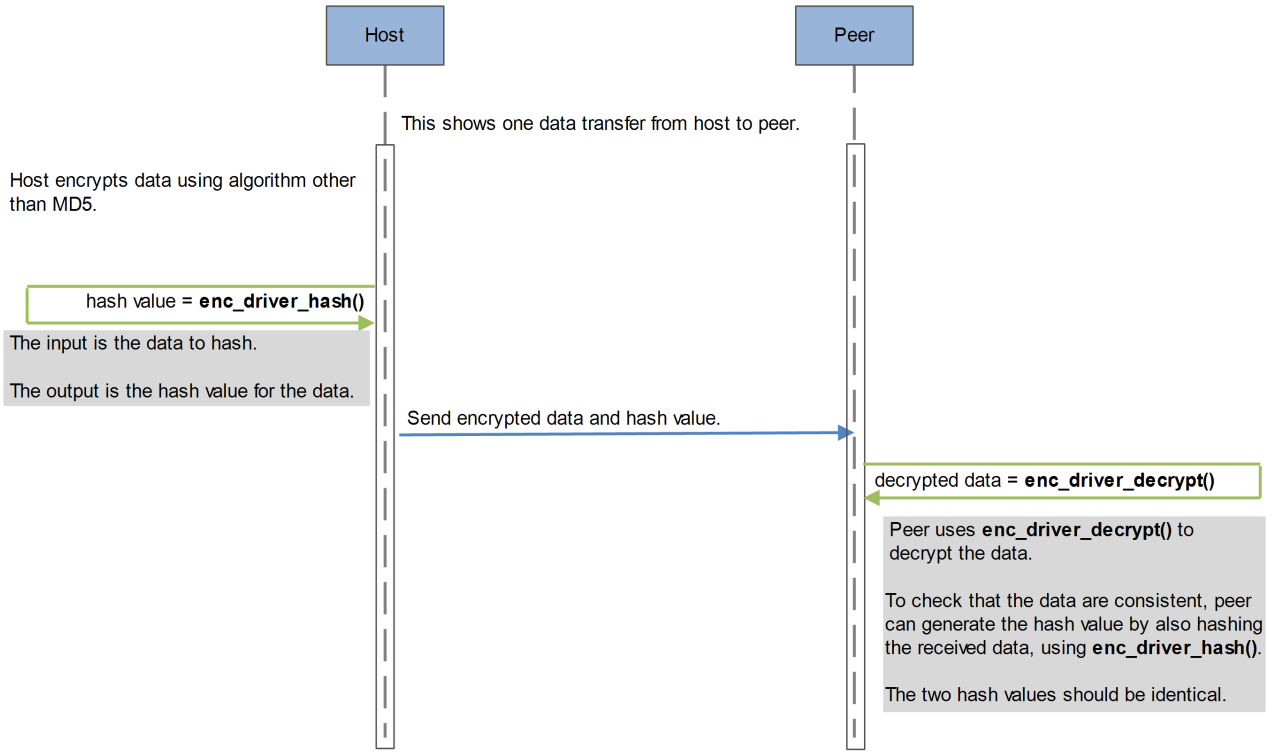
p_in[] points to the data to be hashed.

The output data is the calculated hash value, stored in *p_out[]*. The output length must be set to the output buffer size. It must be at least 16 bytes.

This function is stateful. The final digest is calculated only if the output buffer is defined. When the final digest is calculated the internal state machine is reset and calculation can be performed on new data.

Sequence Diagram

The following sequence diagram shows the process for MD5:



HMAC-MD5 Usage

This driver uses MD5 hashing inside HMAC. The EEM function **enc_driver_encrypt()** is used to calculate the hash value of the given data.

p_in[] points to the data to be hashed.

In this case the relevant parts of the *t_enc_cypher_data* structure are as follows:

Element	Type	Description
p_ecd_key	void *	A pointer to the buffer storing the HMAC key.
ecd_key_size	uint16_t	The length of the key in bytes. This is from 1 to 63.

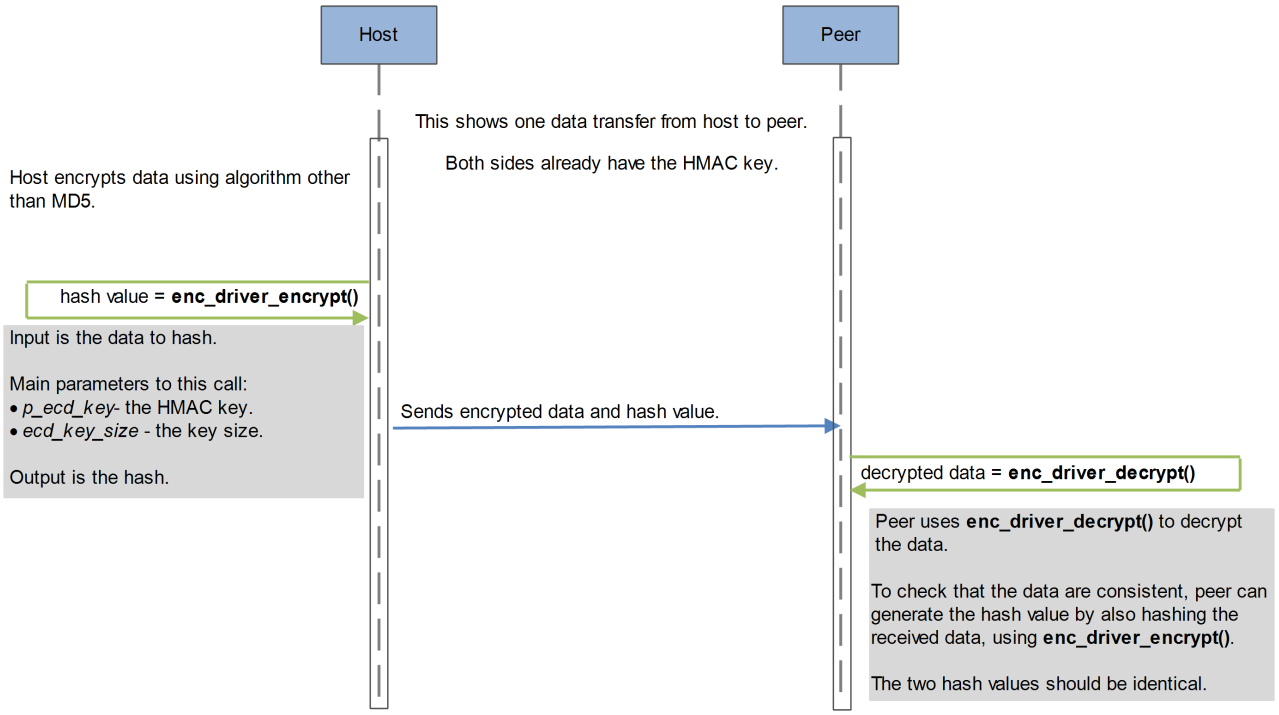
Other fields are discarded but should be set to NULL.

The output data is the calculated hash value, stored in *p_out[]*. The output length must be set to the output buffer size. It must be at least 16 bytes.

This function is stateful. The final digest is calculated only if the output buffer is defined. When the final digest is calculated the internal state machine is reset and calculation can be performed on new data.

Sequence Diagram

The following sequence diagram shows the process for HMAC-MD5:



HMAC-MD5-96 Usage

This driver uses MD5 hashing inside HMAC. The EEM function **enc_driver_encrypt()** is used as for HMAC-MD5. The only difference is that it truncates the output to 12 bytes.

Using the Module

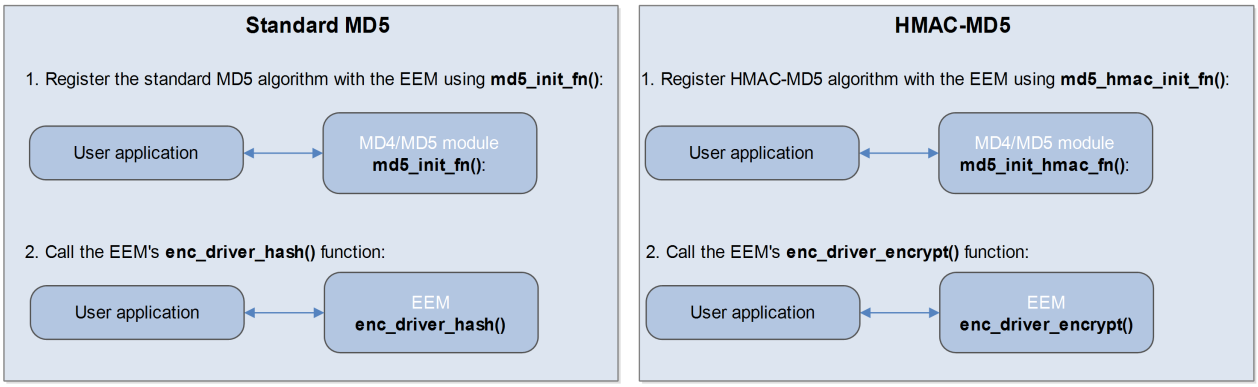
To use any of these algorithms:

1. Register the algorithm(s) with HCC's Embedded Encryption Manager (EEM).
2. Access the algorithm by using standard EEM calls, described in the HCC Embedded Encryption Manager User Guide.

- For standard MD4 or MD5, use the EEM's **enc_driver_hash()** function.
- For the HMAC variants, use the **enc_driver_encrypt()** function (this is because HMAC requires a key to be passed to the algorithm).

Registering an algorithm with the EEM makes it usable by other applications (for example, HCC's TLS /DTLS) through a standard interface. The EEM is the core component of HCC's encryption system.

The process is shown below for two of the algorithms:



A complete test suite is included for validating all of the algorithms.

Note:

- Although every attempt has been made to simplify the system's use, to get the best results you must understand clearly the requirements of the systems you design.
- HCC Embedded offers hardware and firmware development consultancy to help you implement your system; contact sales@hcc-embedded.com.

1.2 Feature Check

The main features of the MD5 module are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Conforms to the HCC Coding Standard including full MISRA compliance.
- Designed for integration with both RTOS and non-RTOS based systems.
- Conforms to the HCC Embedded Encryption Manager (EEM) standard and is compatible with the EEM.
- Supports MD4 (RFC 1320, RFC 6150), MD5 (RFC 1321, RFC 6151), HMAC-MD5 (RFC 6151) and HMAC-MD5-96. The HMAC implementation is compliant with RFC 2104.
- Integral test suite gives complete logical coverage test of each algorithm.

1.3 Packages and Documents

Packages

The table below lists the packages that you need in order to use this module.

Package	Description
hcc_base_docs	This contains the two guides that will help you get started.
enc_base	The EEM base package.
enc_md5	The MD5 package described in this document.

Documents

For an overview of HCC verifiable embedded network encryption, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC Embedded Encryption Manager User Guide

This document describes the EEM.

HCC Encryption Test Suite User Guide

This document describes how to run tests to validate the algorithms.

HCC Message Digest Algorithm 5 User Guide

This is this document.

1.4 Change History

This section describes past changes to this manual.

- To view or download earlier manuals, see Encryption PDFs.
- For the history of changes made to the package code itself, see History: enc_md5.

The current version of this manual is 1.50 BETA. The previous versions are as follows:

Manual version	Date	Software version	Reason for change
1.50B	2018-02-22	1.10	Extended *Introduction*, added new test options and function.
1.40B	2017-06-15	1.09	New *Change History* format.
1.30B	2017-05-23	1.09	Added support for MD4.
1.20B	2017-01-11	1.07	Added lists of functions to API headers.
1.10B	2016-03-09	1.04	Added *Change History*.
1.00B	2015-02-11	1.03	First online version.

2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any file except the configuration file.

2.1 API Header File

The file **src/api/api_enc_sw_md5.h** is the only file that should be included by an application using this module. It defines the Application Programming Interface (API) functions.

2.2 Configuration File

The file **src/config/config_enc_sw_md5.h** contains all the configurable parameters of the system. Configure these as required. This is the only file in the module that you should modify.

2.3 System File

The file **src/enc/software/md5/md5.c** contains the source code. **This file should only be modified by HCC**.

2.4 Test File

The file **src/enc/test/test_md5.c** contains the test registration source code. **This file should only be modified by HCC**.

2.5 Version File

The file **src/version/ver_enc_sw_md5.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

3 Configuration Options

Set the system configuration options in the file **src/config/config_enc_sw_md5.h**. This section lists the available configuration options and their default values.

MD5_INSTANCE_NR

The number of allowed MD5 stateful instances. The default is 1. Set this to 0 to disable the driver.

MD4_INSTANCE_NR

The number of allowed MD4 stateful instances. The default is 1. Set this to 0 to disable the driver.

MD5_HMAC_INSTANCE_NR

The number of allowed HMAC-MD5 stateful instances. The default is 1. Set this to 0 to disable the driver.

MD5_HMAC96_INSTANCE_NR

The number of allowed HMAC-MD5-96 stateful instances. The default is 1. Set this to 0 to disable the driver.

MD5_TEST_ENABLE

Keep the default of 1 to enable the MD5/MD4 test suite. Otherwise, set this to 0.

The following options set the MD5 tests' initialization functions; redefine these if you want to use another set of drivers for a compatibility check.

MD5_TEST_MD5_INITFN

The MD5 hash driver initialization function. The default is *&md5_init_fn*.

MD5_TEST_MD4_INITFN

The MD4 hash driver initialization function. The default is *&md4_init_fn*.

MD5_TEST_MD5_HMAC96_INITFN

The HMAC-MD5-96 driver initialization function. The default is *&md5_hmac96_init_fn*.

MD5_TEST_MD5_HMAC_INITFN

The HMAC-MD5 driver initialization function. The default is *&md5_hmac_init_fn*.

4 Application Programming Interface

This section describes the Application Programming Interface (API) functions, the hash output size, key lengths, and the error codes.

4.1 Functions

Call the initialization functions from the EEM to register the algorithms with it. Call the test function to register the MD5/MD4 tests with the EEM test module.

Once algorithms are registered with the EEM, call standard EEM functions to use them, as follows:

- For standard MD4 or MD5, use the **enc_driver_hash()** function.
- For the HMAC variants, use the **enc_driver_encrypt()** function (this is because HMAC requires a key to be passed to the algorithm).

The functions are the following:

Function	Description
md4_init_fn()	Called from the EEM, this registers the MD4 algorithm with it.
md5_init_fn()	Called from the EEM, this registers the MD5 algorithm with it.
md5_hmac_init_fn()	Called from the EEM, this registers the HMAC-MD5 algorithm with it.
md5_hmac96_init_fn()	Called from the EEM, this registers the HMAC-MD5-96 algorithm with it.
md5_register_tests()	Registers the MD5/MD4 tests with the EEM test module.

md4_init_fn

Call this initialization function from the EEM to register the MD4 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing MD4 functions to the EEM. This structure is described in the the *HCC Embedded Encryption Manager User Guide*.

> **Note:** This implementation is stateful so a hash can be calculated from multiple data buffer inputs (partial data). To calculate a hash from partial data, set the output buffer to NULL.

Format

```
t_enc_ret md4_init_fn ( t_enc_driver_fn const * * const pp_encdriver )
```

Arguments

Parameter	Description	Type
pp_encdriver	A pointer to a *t_enc_driver_fn* structure containing MD4 functions.	t_enc_driver_fn * *

Return Values

Return value	Description
ENC_SUCCESS	Successful execution.
ENC_INVALID_ERR	The module has already been initialized.

md5_init_fn

Call this initialization function from the EEM to register the MD5 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing MD5 functions to the EEM. This structure is described in the the *HCC Embedded Encryption Manager User Guide*.

> **Note:** This implementation is stateful so a hash can be calculated from multiple data buffer inputs (partial data). To calculate a hash from partial data, set the output buffer to NULL.

Format

```
t_enc_ret md5_init_fn ( t_enc_driver_fn const * * const pp_encdriver )
```

Arguments

Parameter	Description	Type
pp_encdriver	A pointer to a *t_enc_driver_fn* structure containing MD5 functions.	t_enc_driver_fn * *

Return Values

Return value	Description
ENC_SUCCESS	Successful execution.
ENC_INVALID_ERR	The module has already been initialized.

md5_hmac_init_fn

Call this initialization function from the EEM to register the HMAC-MD5 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing HMAC-MD5 functions to the EEM. This structure is described in the the *HCC Embedded Encryption Manager User Guide*.

This algorithm calculates the HMAC-MD5. The encryption function calculates the HMAC value. The HMAC output length is equal to MD5_HMAC_OUT_LEN.

> **Note:** This implementation is stateful so a hash can be calculated from multiple data buffer inputs (partial data). To calculate a hash from partial data, set the output buffer to NULL.

Format

```
t_enc_ret md5_hmac_init_fn ( t_enc_driver_fn const * * const pp_encdriver )
```

Arguments

Parameter	Description	Type
pp_encdriver	A pointer to a *t_enc_driver_fn* structure containing HMAC-MD5 functions.	t_enc_driver_fn * *

Return Values

Return value	Description
ENC_SUCCESS	Successful execution.
ENC_INVALID_ERR	The module has already been initialized.

md5_hmac96_init_fn

Call this initialization function from the EEM to register the HMAC-MD5-96 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing HMAC-MD5-96 functions to the EEM. This structure is described in the the *HCC Embedded Encryption Manager User Guide*.

This algorithm calculates the HMAC-MD5-96 value. The encryption function calculates the HMAC value. The HMAC output length is equal to MD5_HMAC_96_OUT_LEN.

> **Note:** This implementation is stateful so a hash can be calculated from multiple data buffer inputs (partial data). To calculate a hash from partial data, set the output buffer to NULL.

Format

```
t_enc_ret md5_hmac96_init_fn ( t_enc_driver_fn const * * const pp_encdriver )
```

Arguments

Parameter	Description	Type
pp_encdriver	A pointer to a *t_enc_driver_fn* structure containing HMAC-MD5-96 functions.	t_enc_driver_fn * *

Return Values

Return value	Description
ENC_SUCCESS	Successful execution.
ENC_INVALID_ERR	The module has already been initialized.

md5_register_tests

Call this function to register the MD5/MD4 tests with the EEM test module.

Once you have registered the tests, you can execute the test suite as directed in the *HCC Encryption Test Suite User Guide*.

> **Note:** The MD5_TEST_ENABLE configuration option must be set to 1 to enable this function.

Format

```
t_enc_ret md5_register_tests ( void )
```

Arguments

None.

Return Values

Return value	Description
ENC_SUCCESS	Successful execution.
Else	See Error Codes.

4.2 Hash Output Sizes

The hash output sizes are defined in the file **src/api/api_enc_sw_md5.h**.

Name	Value	Description
MD4_OUT_LEN	16	The size of the MD4 hash output in bytes.
MD5_OUT_LEN	16	The size of the Md5 hash output in bytes.

4.3 Key Lengths

The key lengths are as follows:

Name	Value	Description
MD5_HMAC_96_KEY_LEN	16	The length of the MD5-HMAC-96 key value.
MD5_HMAC_96_OUT_LEN	12	The length of the MD5-HMAC-96 MAC value.
MD5_HMAC_OUT_LEN	16	The length of the MD5-HMAC MAC value.

4.4 Error Codes

The table below lists the error codes that may be generated by the API calls.

Error code	Value	Meaning
ENC_SUCCESS	0	Successful execution.
ENC_INVALID_ERR	1	The module has already been initialized.

5 Integration

The MD5 module is designed to be as open and as portable as possible. No assumptions are made about the functionality, the behavior, or even the existence, of the underlying operating system. For the system to work at its best, perform the porting outlined below. This is a straightforward task for an experienced engineer.

5.1 OS Abstraction Layer

The module uses the OS Abstraction Layer (OAL) that allows it to run seamlessly with a wide variety of RTOSes, or without an RTOS.

The module uses the following OAL components:

OAL Resource	Number Required
Tasks	0
Mutexes	1 per algorithm enabled
Events	0

5.2 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer. For full details of these elements, see the *HCC Base Platform Support Package User Guide*.

The module makes use of the following standard PSP functions:

Function	Package	Element	Description
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.

The module makes use of the following standard PSP macros:

Macro	Package	Element	Description
PSP_RD_LE32	psp_base	psp_endianness	Reads a 32 bit value stored as little-endian from a memory location.
PSP_WR_LE32	psp_base	psp_endianness	Writes a 32 bit value to be stored as little-endian to a memory location.