Tiger Hash Algorithm User Guide

Version 1.00 BETA

For use with Tiger Hash Algorithm module versions 1.03 and above

Date: 22-Feb-2018 10:42

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 any of the following Tiger hash algorithms: Tiger 128, Tiger 160, Tiger 192, and Tiger 192 HMAC. (HMAC stands for Hash Message Authentication Code.)

Note: The Tiger hash algorithms are rarely used but they are needed because they may be used by Internet Key Exchange (IKE).

Overview

The Tiger 128, Tiger 160, and Tiger 192 algorithms have output lengths of 16, 20, and 24 bytes, respectively.

Tiger is not stateful. You call it with data and it returns the resultant hash value.

Tiger was invented to be much faster than other hash algorithms but this only applies where a 64 bit calculator is available. Tiger uses large translation tables so needs 8 KBytes of ROM.

Tiger 128

This driver implements the TIGER 128 hashing algorithm.

The EEM function **enc_driver_hash()** is used to calculate the hash value of the given data. The function is not stateful.

p_in[] points to the data to be hashed. The length of the data (*in_len*) does not need to be aligned.

The output data from **enc_driver_hash()** is the hash value, stored in *p_out[]*.

The output length, *p_out_len*, must be set to the output buffer size. This must be at least 16 bytes.

Tiger 160

This driver implements the TIGER 160 hashing algorithm.

The EEM function **enc_driver_hash()** is used to calculate the hash value of the given data. The function is not stateful.

p_in[] points to the data to be hashed. The length of the data (*in_len*) does not need to be aligned.

The output data from **enc_driver_hash()** is the hash value, stored in *p_out[]*.

The output length, *p_out_len*, must be set to the output buffer size. This must be at least 20 bytes.

Tiger 192

This driver implements the TIGER 192 hashing algorithm.

The EEM function **enc_driver_hash()** is used to calculate the hash value of the given data. The function is not stateful.

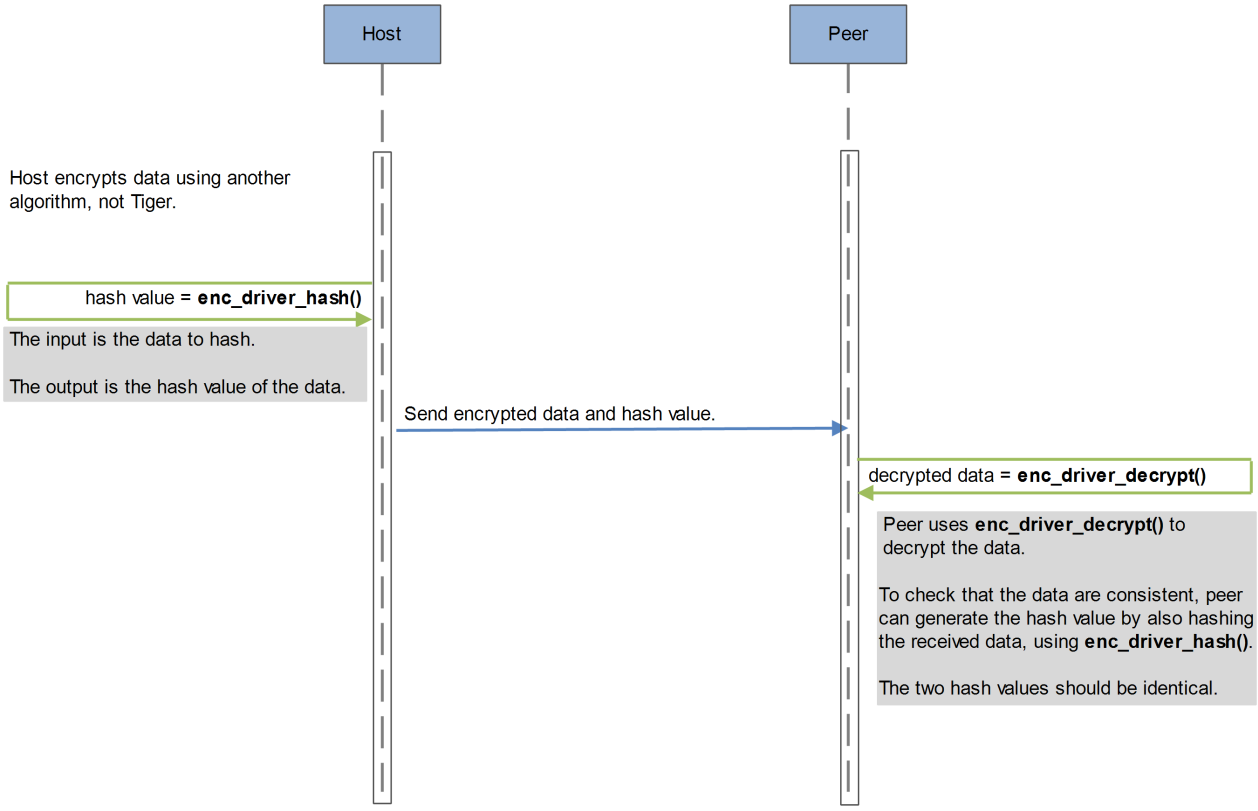
p_in[] points to the data to be hashed. The length of the data (*in_len*) does not need to be aligned.

The output data from **enc_driver_hash()** is the hash value, stored in *p_out[]*.

The output length, *p_out_len*, must be set to the output buffer size. This must be at least 24 bytes.

Sequence Diagram

This diagram shows the sequence used for a single transfer between host and peer:



Tiger 192 HMAC

This driver implements HMAC with the TIGER 192 hashing algorithm.

The EEM function **enc_driver_encrypt()** is used to calculate the hash value of the given data. The function is not stateful.

p_in[] points to the data to be hashed. The length of the data (*in_len*) does not need to be aligned.

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; this must be greater than 0.

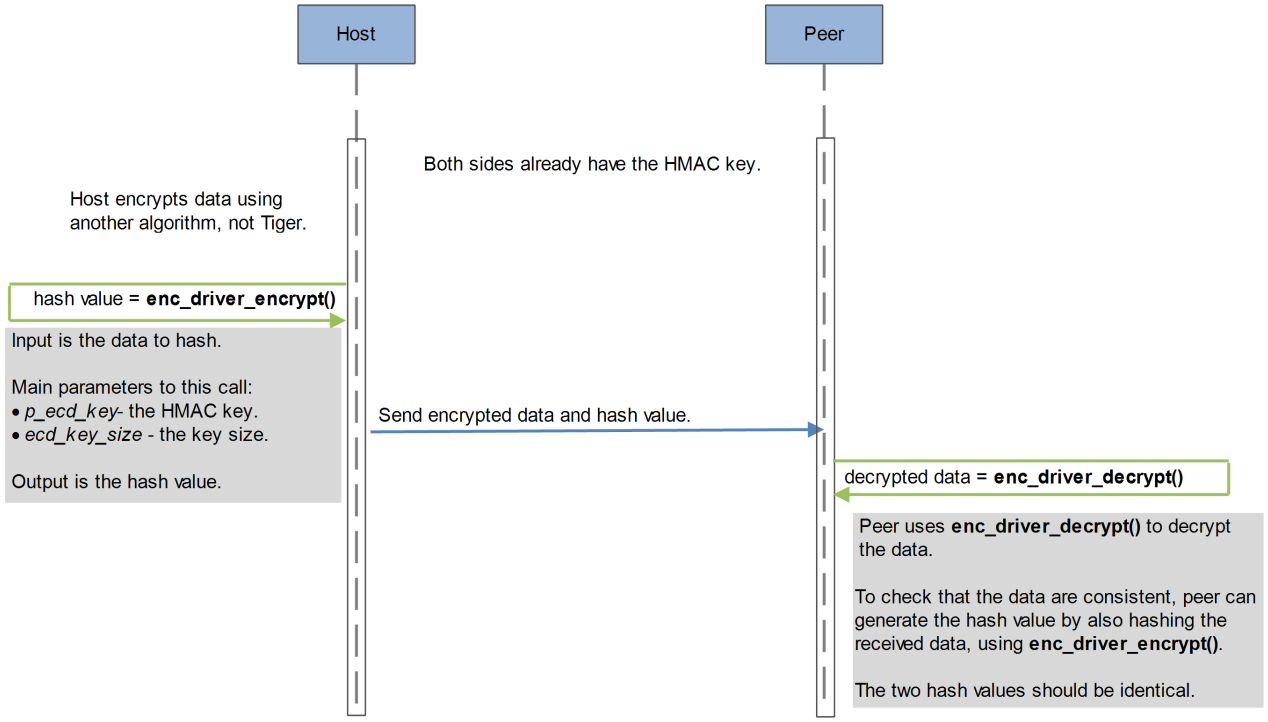
Other fields are discarded but should be set to NULL.

The output data from **enc_driver_encrypt()** is the hash value, stored in *p_out[]*.

The output length, *p_out_len*, must be set to the output buffer size. This must be at least 24 bytes.

Sequence Diagram

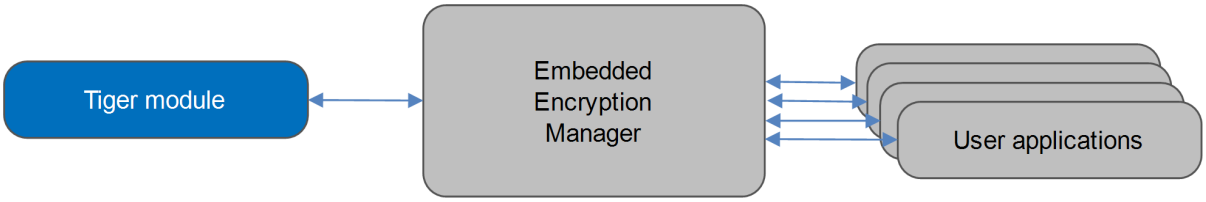
This diagram shows the sequence used for a single transfer between host and peer:



Using the Module

You register the Tiger module with HCC's Embedded Encryption Manager (EEM), making 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 system structure is shown below:



A complete test suite is included for validating the TIGER 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 Tiger Hash Algorithm 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 Tiger 128, Tiger 160, Tiger 192, and Tiger 192 HMAC implementations.
- 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_tiger	The Tiger 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 Embedded Encryption Test Suite User Guide

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

HCC Tiger Hash Algorithm User Guide

This is this document.

1.4 Change History

This section describes past changes to this manual.

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

The current version of this manual is 1.00 BETA.

Manual version	Date	Software version	Reason for change
1.00B	2018-02-22	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 files except the configuration file.

2.1 API Header File

The file **src/api/api_enc_sw_tiger.h** is the only file that should be included by an application using this module. For details of the functions, see Application Programming Interface.

2.2 Configuration File

The file **src/config/config_enc_sw_tiger.h** contains 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/tiger/tiger.c** contains the source code. **This file should only be modified by HCC**.

2.4 Test File

The file **src/enc/test/test_tiger.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_tiger.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_tiger.h**.

TIGER_TEST_ENABLE

Keep the default of 1 to enable the Tiger test suite. Otherwise, set it to 0.

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

TIGER_TEST_TIGER_INITFN

The TIGER hash driver init function. The default is *&tiger192_init_fn*.

TIGER_TEST_TIGER160_INITFN

The TIGER 160 hash driver init function. The default is *&tiger160_init_fn*.

TIGER_TEST_TIGER128_INITFN

The TIGER 128 hash driver init function. The default is *&tiger128_init_fn*.

TIGER_TEST_TIGER_HMAC_INITFN

The TIGER 192 HMAC driver init function. The default is *&tiger192_hmac_init_fn*.

4 Application Programming Interface

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

4.1 Functions

These are the functions that you call from the EEM to register the algorithms with it. Once algorithms are registered with the EEM, call standard EEM functions to use them, as follows:

- For Tiger 128, 160 or 192, use the **enc_driver_hash()** function.
- For Tiger 192 HMAC, 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
tiger128_init_fn()	Called from the EEM, this registers the Tiger 128 algorithm with it.
tiger160_init_fn()	Called from the EEM, this registers the Tiger 160 algorithm with it.
tiger192_init_fn()	Called from the EEM, this registers the Tiger 192 algorithm with it.
tiger192_hmac_init_fn()	Called from the EEM, this registers the Tiger 192 HMAC algorithm with it.
tiger_register_tests()	Registers the TIGER tests with the EEM test module.

tiger128_init_fn

Call this initialization function from the EEM to register the Tiger 128 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing Tiger 128 functions to the EEM. This structure is described in the the *HCC Embedded Encryption Manager User Guide*.

Format

```
t_enc_ret tiger128_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 Tiger 128 functions.	t_enc_driver_fn * *

Return Values

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

tiger160_init_fn

Call this initialization function from the EEM to register the Tiger 160 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing Tiger 160 functions to the EEM. This structure is described in the the *HCC Embedded Encryption Manager User Guide*.

Format

```
t_enc_ret tiger160_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 Tiger 160 functions.	t_enc_driver_fn * *

Return Values

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

tiger192_init_fn

Call this initialization function from the EEM to register the Tiger 192 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing Tiger 192 functions to the EEM. This structure is described in the the *HCC Embedded Encryption Manager User Guide*.

Format

```
t_enc_ret tiger192_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 Tiger 192 functions.	t_enc_driver_fn * *

Return Values

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

tiger192_hmac_init_fn

Call this initialization function from the EEM to register the Tiger 192 HMAC algorithm with it.

This forwards the *t_enc_driver_fn* structure containing Tiger 192 HMAC functions to the EEM. This structure is described in the the *HCC Embedded Encryption Manager User Guide*.

Format

```
t_enc_ret tiger192_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 Tiger 192 HMAC functions.	t_enc_driver_fn * *

Return Values

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

tiger_register_tests

Call this function to register the TIGER 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 TIGER_TEST_ENABLE configuration option must be set to 1 to enable this function.

Format

```
t_enc_ret tiger_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_tiger.h**.

Name	Value	Description
TIGER_BLOCK_LEN	20	The size of the Tiger message block in bytes.
TIGER192_HMAC_KEY_LEN	20	The preferred length of the Tiger 192 HMAC key value.
TIGER128_OUT_LEN	24	The size of the Tiger 128 hash output.
TIGER160_OUT_LEN	20	The size of the Tiger 160 hash output.
TIGER192_OUT_LEN	16	The size of the Tiger 192 hash output.
TIGER192_HMAC_OUT_LEN	20	The size of the Tiger 192 HMAC MAC value.

4.3 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 Tiger 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 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. These are defined in the file **psp/include /psp_endianness.h**.

Macro	Package	Element	Description
PSP_WR_LE32	psp_base	psp_endianness	Writes a 32 bit value stored as little-endian to a memory location.
PSP_RD_LE64	psp_base	psp_endianness	Reads a 64 bit value stored as little-endian from a memory location.
PSP_WR_LE64	psp_base	psp_endianness	Writes a 64 bit value stored as little-endian to a memory location.

Note: You must modify this PSP implementation for your specific microcontroller and development board.