Secure Hash Algorithm (SHA) User Guide

BETA DRAFT

Version 1.40 BETA

For use with SHA module versions 1.09 and above

Exported on 08/14/2018

Table of Contents

Version 1.40 BETA

For use with SHA module versions 1.09 and above

Tip: Use the Contents list on the left to navigate within this manual.

Encryption Documents Home

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 supported Secure Hash Algorithms (SHAs).

Overview

The SHA module implements the following Secure Hash Algorithms: SHA-1, SHA-1 HMAC, SHA1-HMAC-96, SHA-256, SHA-384 or SHA-512. (HMAC stands for Hash Message Authentication Code.) This module is part of the CryptoCore™ Pro security suite.

IPsec needs HMAC variants but only uses SHA1-HMAC. Modern protocols like TLS use their own HMAC-type variant so just use a standard hash.

If the output buffer *outbuf* is set to NULL, this means more data is to come. If *outbuf* is present the final result is returned and the next call starts.

The different algorithms have different output lengths, as follows:

- SHA-1 - 20 bytes.
- SHA-256 - 32 bytes.
- SHA-384 - 48 bytes.
- SHA-512 - 64 bytes.
- SHA1-HMAC - 20 bytes.
- SHA1-HMAC-96 - 12 bytes.

The HMAC variants allow a key to be added to the hash calculation

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

- For standard SHA-xxx, use the EEM's **enc_driver_hash()** function.
- For the HMAC variants, use the EEM's **enc_driver_encrypt()** and **enc_driver_decrypt()** functions. This is because HMAC requires a key to be passed to the algorithm.

Note: SHA1 is considered weak and is rarely used now.

SHA-256, SHA-384 and SHA-512

For these three algorithms the EEM function **enc_driver_hash()** is used to calculate the hash value of given data. *p_in[]* points to the data to be hashed.

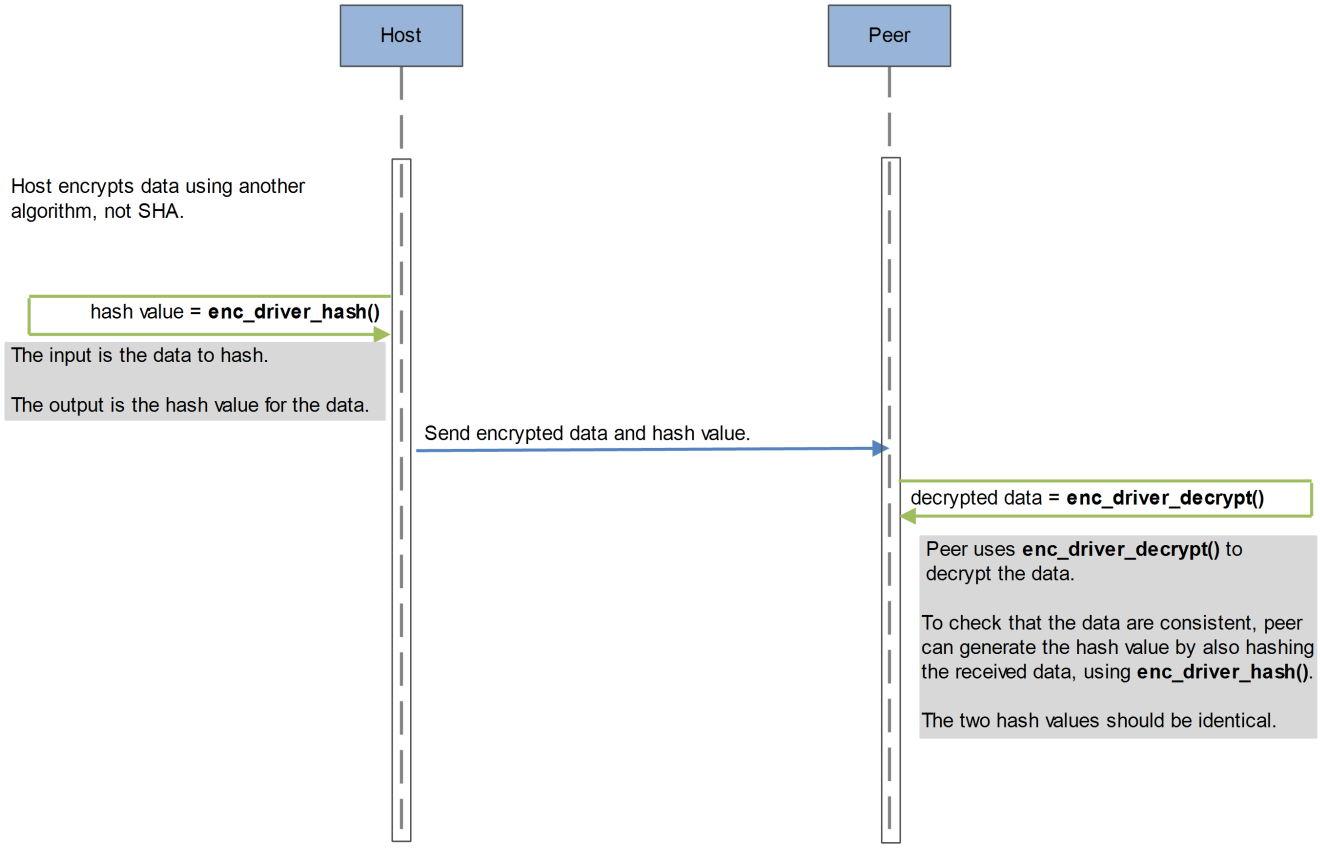
The output data from **enc_driver_hash()** is the hash value, stored in *p_out[]*.

The output length, *p_out_len*, must be set to the output buffer size. This must be at least 32 bytes for SHA-256, at least 48 bytes for SHA-384 and at least 64 bytes for SHA-512.

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

Sequence Diagram

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



SHA1-HMAC

This driver uses SHA-1 hashing within HMAC.

For this algorithm the EEM function **enc_driver_encrypt()** is used to calculate the hash value of 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.

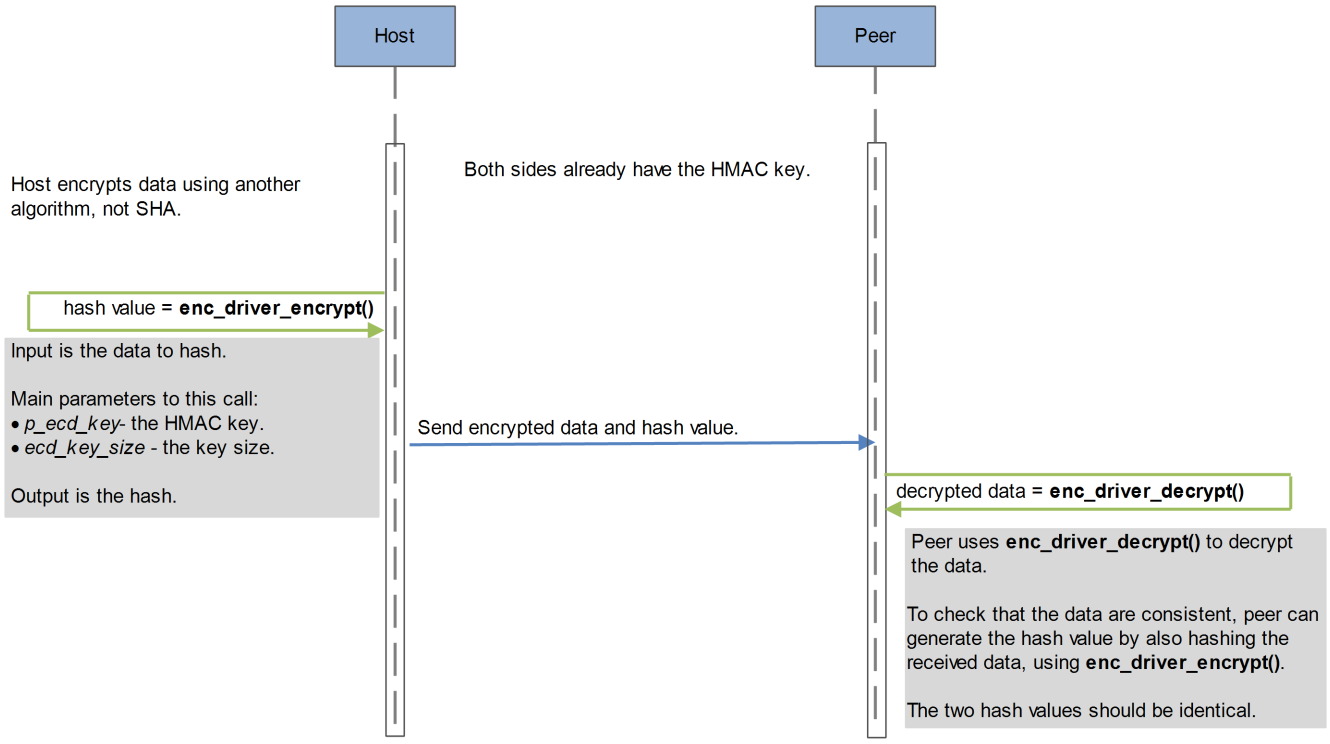
The output data from **enc_driver_decrypt()** is the hash value, stored in *p_out[]*.

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

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

Sequence Diagram

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



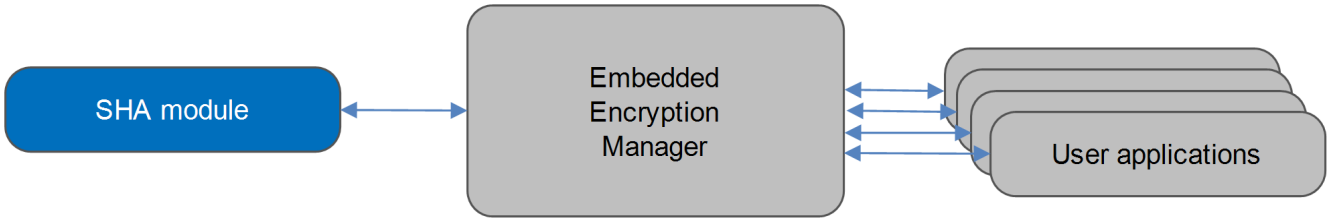
SHA1-HMAC-96

This driver uses SHA-1 hashing within HMAC. This algorithm operates exactly the same as SHA1-HMAC except that the output is truncated to 12 bytes.

Using the Module

You register the SHA 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 all of the HCC 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 SHA 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.
- The SHA-1 implementation conforms to RFC 3174.
- The SHA-256, SHA-384, and SHA-512 implementations conform to RFC 4634.
- The SHA-HMAC-96 implementation conforms to RFC 2404.
- 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_sha	The SHA 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 Secure Hash Algorithm (SHA) 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_sha.

The current version of this manual is 1.40 BETA. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.40B	2018-02-22	1.09	Extended *Introduction*, added new test options and function.
1.30B	2017-06-15	1.08	New *Change History* format.
1.20B	2017-01-11	1.07	Added lists of functions to API headers.
1.10B	2016-03-09	1.03	Added Change History, SHA-512 and SHA-384.
1.00B	2015-02-11	1.02	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_sha.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_sha.h** contains all the configurable parameters of the system. Configure these as required. For details of these options, see Configuration Options.

2.3 System File

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

2.4 Test File

The file **src/enc/test/test_sha.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_sha.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_sha.h**. This section lists the available options and their default values.

SHA1_INSTANCE_NR

The number of allowed SHA-1 stateful instances. The default is 1. Set this to 0 to disable SHA-1.

SHA1_HMAC_INSTANCE_NR

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

SHA1_HMAC_96_INSTANCE_NR

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

SHA256_INSTANCE_NR

The number of allowed SHA-256 stateful instances. The default is 1. Set this to 0 to disable SHA-256.

SHA384_INSTANCE_NR

The number of allowed SHA-384 stateful instances. The default is 1. Set this to 0 to disable SHA-384.

SHA512_INSTANCE_NR

The number of allowed SHA-512 stateful instances. The default is 1. Set this to 0 to disable SHA-512.

3.1 Test Options

SHA_TEST_ENABLE

Keep the default of 1 to enable the SHA test suite. Otherwise, set this to 0.

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

SHA_TEST_SHA512_INITFN

The SHA-512 hash driver initialization function. The default is *&sha512_init_fn*.

SHA_TEST_SHA384_INITFN

The SHA-384 hash driver initialization function. The default is *&sha384_init_fn*.

SHA_TEST_SHA256_INITFN

The SHA-256 hash driver initialization function. The default is *&sha256_init_fn*.

SHA_TEST_SHA1_INITFN

The SHA-1 hash driver initialization function. The default is *&sha1_init_fn*.

SHA_TEST_SHA1_HMAC96_INITFN

The SHA1-HMAC-96 hash driver initialization function. The default is *&sha1_hmac_96_init_fn*.

SHA_TEST_SHA1_HMAC_INITFN

The SHA1-HMAC hash driver initialization function. The default is *&sha1_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

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

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

- For standard SHA-1, SHA-256, SHA-384 and SHA-512, 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
sha1_init_fn()	Called from the EEM, this registers the SHA-1 algorithm with it.
sha1_hmac_init_fn()	Called from the EEM, this registers the SHA1-HMAC algorithm with it.
sha1_hmac_96_init_fn()	Called from the EEM, this registers the SHA1-HMAC-96 algorithm with it.
sha256_init_fn()	Called from the EEM, this registers the SHA-256 algorithm with it.
sha384_init_fn()	Called from the EEM, this registers the SHA-384 algorithm with it.
sha512_init_fn()	Called from the EEM, this registers the SHA-512 algorithm with it.
sha_register_tests()	Registers the SHA tests with the EEM test module.

sha1_init_fn

Call this initialization function from the EEM to register the SHA-1 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing SHA-1 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 sha1_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 SHA-1 functions.	t_enc_driver_fn * *

Return Values

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

sha1_hmac_init_fn

Call this initialization function from the EEM to register the SHA1-HMAC algorithm with it.

This forwards the *t_enc_driver_fn* structure containing SHA1-HMAC 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 sha1_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 SHA1-HMAC functions.	t_enc_driver_fn * *

Return Values

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

sha1_hmac_96_init_fn

Call this initialization function from the EEM to register the SHA1-HMAC-96 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing SHA1-HMAC-96 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 sha1_hmac_96_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 SHA1-HMAC-96 functions.	t_enc_driver_fn * *

Return Values

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

sha256_init_fn

Call this initialization function from the EEM to register the SHA-256 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing SHA-256 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 sha256_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 SHA-256 functions.	t_enc_driver_fn * *

Return Values

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

sha384_init_fn

Call this initialization function from the EEM to register the SHA-384 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing SHA-384 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 sha384_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 SHA-384 functions.	t_enc_driver_fn * *

Return Values

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

sha512_init_fn

Call this initialization function from the EEM to register the SHA-512 algorithm with it.

This forwards the *t_enc_driver_fn* structure containing SHA-512 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 sha512_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 SHA-512 functions.	t_enc_driver_fn * *

Return Values

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

sha_register_tests

Call this function to register the SHA 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 SHA_TEST_ENABLE configuration option must be set to 1 to enable this function.

Format

```
t_enc_ret sha_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_sha.h**.

Name	Value	Description
SHA1_OUT_LEN	20	The size of the SHA-1 hash output.
SHA1_HMAC_OUT_LEN	20	The size of the SHA1 HMAC MAC value.
SHA1_HMAC_96_OUT_LEN	12	The size of the SHA1 HMAC_96 MAC value.
SHA256_OUT_LEN	32	The size of the SHA-256 hash output.
SHA384_OUT_LEN	48	The size of the SHA-384 hash output.
SHA512_OUT_LEN	64	The size of the SHA-512 hash output.

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 SHA 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 used
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. These are defined in the file **psp/include/psp_endianness.h**.

Macro	Package	Element	Description
PSP_RD_BE32	psp_base	psp_endianness	Reads a 32 bit value stored as big-endian from a memory location.
PSP_WR_BE32	psp_base	psp_endianness	Writes a 32 bit value stored as big-endian to a memory location.
PSP_RD_BE64	psp_base	psp_endianness	Reads a 64 bit value stored as big-endian from a memory location.
PSP_WR_BE64	psp_base	psp_endianness	Writes a 64 bit value stored as big-endian to a memory location.

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