Elliptic Curve Cryptography User Guide

Version 1.00 BETA

For use with Elliptic Curve Cryptography (ECC) module versions 1.07 and above

Date: 22-Feb-2018 10:23

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 use Elliptic Curve Cryptography (ECC) with the following HCC modules:

- Ephemeral Diffie-Hellman (EDH) and Diffie-Hellman (DH) algorithms. When EDH uses ECC it is termed Elliptic Curve Diffie-Hellman (ECDHE). When DH uses ECC it is termed ECDH.
- Digital Signature Standard (DSS) – this uses the Digital Signature Algorithm (DSA). When DSS uses ECC it is termed Elliptic Curve Digital Signature Algorithm (ECDSA).

The ECC module implements both ECDH and ECDSA. ECC allows you to use smaller keys but get the same levels of security.

Overview

ECC is used as the key exchange mechanism by ECDHE/ECHD and the signature algorithm ECDSA. Both ECDH and ECDSA are handled automatically by TLS.

Both mechanisms relay on point multiplication of points that lie on an elliptic curve.

This HCC implementation supports only the following prime field curves:

- SECP160K1
- SECP160R1
- SECP160R2
- SECP192K1
- SECP192R1
- SECP224K1
- SECP224R1
- SECP256K1
- SECP256R1
- SECP384R1
- SECP521R1

SECP256R1 and SECP384R1 are the most commonly used curves and are the curves specified in the *TLS Suite B* cipher suite.

In this module the curves SECP192R1, SECP224R1, SECP256R1, SECP384R1, and SECP521R1 have been optimized by using special reduction functions.

Binary field curves are not supported; these are rarely used in practice.

ECDH

The Elliptic Curve Diffie-Hellman algorithm is used to generate a shared secret key based on host/peer secret values that are not exchanged. The algorithm is very similar to EDH.

Our code calls **psp_random()** to get a random number; see the *Random Number Generation* section below.

Assuming that:

- The peer generates secret value **pa**.
- The host generates secret value **ha**.
- The peer and host negotiate use of curve SECP256R1.

Then:

- The peer calculates SECP256R1 init point***pa** and sends this value to the host.
- The host calculates SECP256R1 init point***ha** and sends this value to the peer.
- Both sides now calculate shared secret SECP256R1 init point***pa***ha**.

ECDH usage

The host has two values:

- Its generated secret value, **ha**.
- The key: the negotiated curve ID.

enc_driver_encrypt()

The EEM function **enc_driver_encrypt()** is used to calculate the public passed value.

p_in[] points to the secret value generated by the host (**ha**). The input size (*in_len*) cannot be longer than the used curve. The maximum input data length for SECP256R1 is 32 bytes.

In this case the relevant part of the *t_enc_cypher_data* structure is as follows:

Element	Type	Description
p_ecd_init_vect	uint8_t *	A pointer to the negotiated curve ID in the format: 0x03U <2 byte curve ID in big-endian> Curve IDs are specified as *t_ecc_named_curve*, according to the NIST specification.

Other fields are discarded but should be set to NULL.

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

enc_driver_decrypt()

The EEM function **enc_driver_decrypt()** is used to calculate the shared secret value.

p_in[] points to the secret value generated by the host (**ha**). The length of the data (*in_len*) does not have to be a multiple of 16 bytes.

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

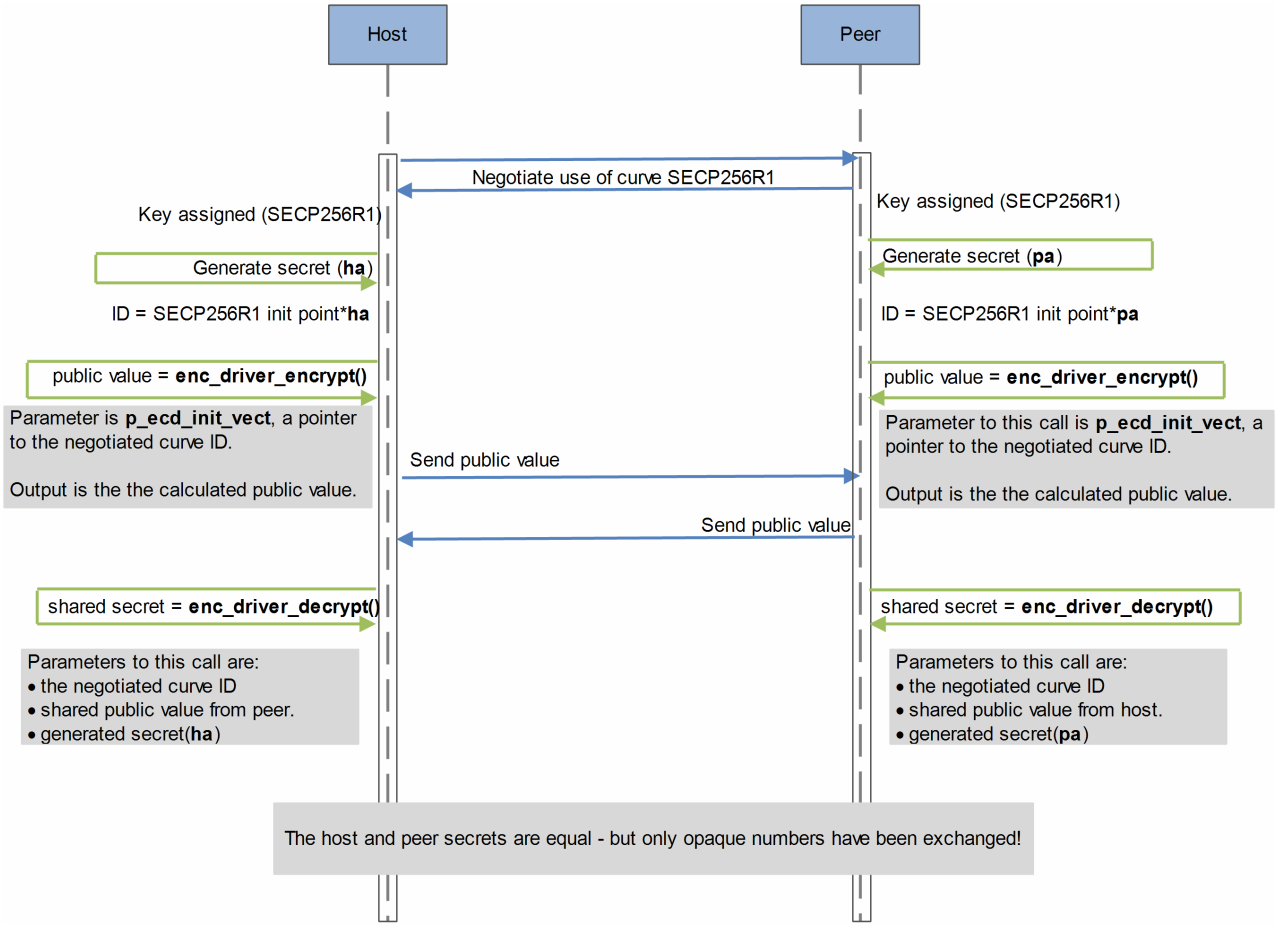
Element	Type	Description
p_ecd_init_vect	uint8_t *	A pointer to the negotiated curve ID.
ecd_init_vect_size	uint16_t	The length of the initialization data vector, always 3.
p_ecd_key	void *	A pointer to the buffer storing the the shared public value received from the peer.

Other fields are discarded but should be set to NULL.

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

Sequence Diagram

The following sequence diagram shows the process:



ECDSA

ECDSA is a signing algorithm. The signature is encoded as two values that represent a random value and the X position of ECC point multiplication.

enc_driver_encrypt()

The EEM function **enc_driver_encrypt()** is used to sign the input data.

p_in[] points to the hash value of the data to be signed. The input size (*in_len*) cannot be longer than the used curve. The maximum input data length for SECP256R1 is 32 bytes.

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 DER-encoded private key for ECDSA, as specified by the X.509 standard.
ecd_key_size	uint16_t	The length of the key in bytes.

Other fields are discarded but should be set to NULL.

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

The output length, *p_out_len*, must be set to the output buffer size. This buffer must be able to store DER-encoded ECC points. In the worst case it is 9 + 2 * curve size in bytes.

enc_driver_decrypt()

The EEM function **enc_driver_decrypt()** is used to check the signature of given data.

p_in[] points to the hash value of the data whose signature is to be checked. The length of the data (*in_len*) must be a multiple of 16 bytes.

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 ECDSA public key (X.509 certificate subject public key information).
ecd_key_size	uint16_t	The length of the public key in bytes.

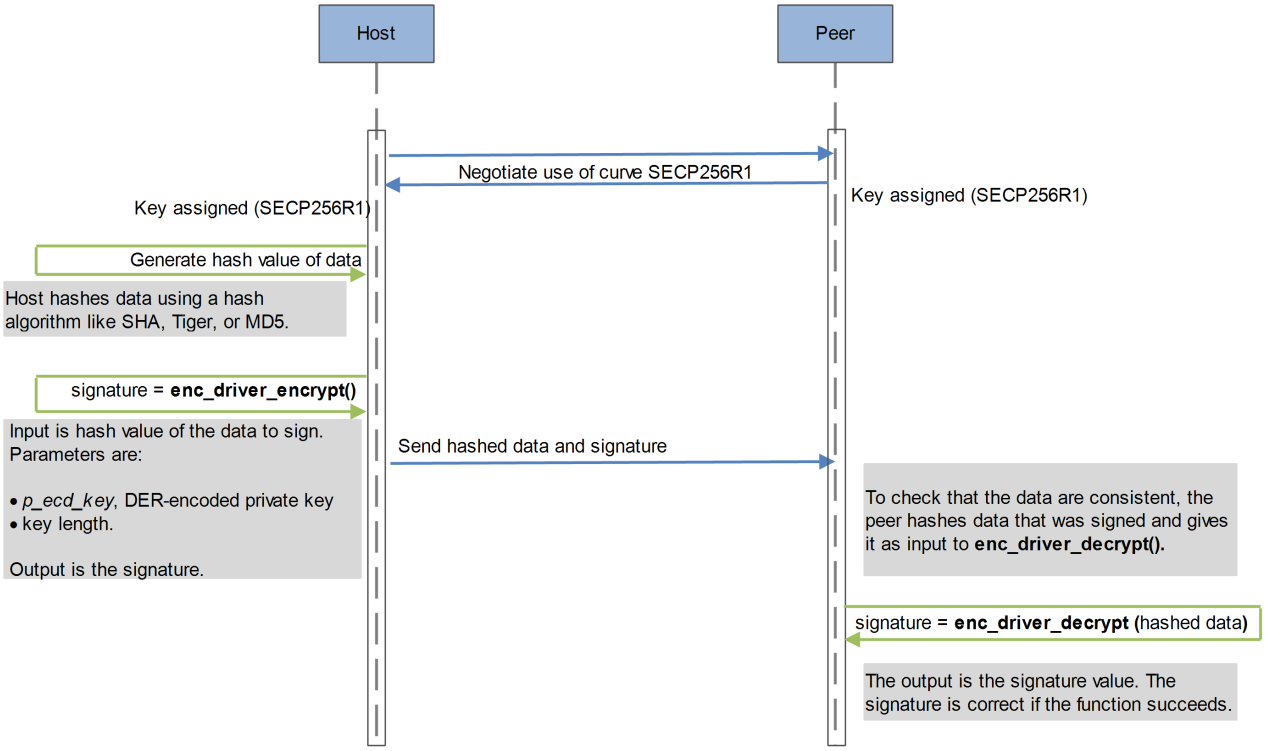
Other fields are discarded but should be set to NULL.

The output data from **enc_driver_decrypt()** is the signature to be checked, stored in *p_out[].*

The output length, *p_out_len*, is set to the length of the signature.

Sequence Diagram

The following sequence diagram shows the process:



Random Number Generation

Random Number Generation (RNG) is critical in security. Random numbers are used as secret values when negotiating keys. If an attacker can predict the secret numbers, it is easy for them to generate the keys.

Use of a True Random Number Generator (TRNG) is recommended. Software implementations can only implement a Pseudo Random Number Generator (PRNG), which may be predicted by an attacker.

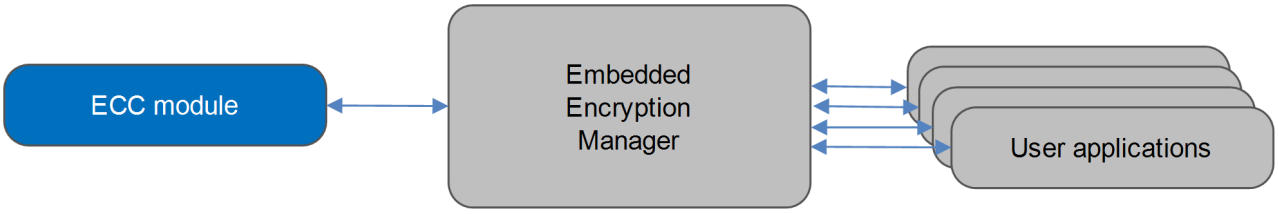
To implement a TRNG, a special hardware module is needed. Most modern chips implement a special module that can be used as a TRNG. If a device does not have a dedicated RND module, it can add an external random generator chip or implement the NIST recommended random generator which uses RealTimeClock: see *ANSI X9.31-1998 Appendix A.2.4*.

HCC provides a simple pseudo-random generator module **psp_getrand()**; port this to use your platform-specific RNG module.

Using the Module

You register the ECC 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 available for validating 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 ECC 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 it.
- Implements both ECDH and ECDSA.
- Allows use of Elliptic Curve Cryptography (ECC) with HCC's Ephemeral Diffie-Hellman (EDH) and Digital Signature Standard (DSS) modules.
- 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_ecc	The ECC package described in this document.
psp_template_base	The base Platform Support Package (PSP).

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 Elliptic Curve Cryptography User Guide

This is this document.

1.4 Change History

To view or download manuals, see Encryption PDFs.

For the history of changes made to the package code itself, see History: enc_ecc.

The current version of this manual is 1.00 BETA.

Manual version	Date	Software version	Reason for change
1.00 BETA	2018-02-22	1.07	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 of these files.

2.1 API Header File

The file **src/api/api_enc_sw_ecc.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_ecc.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 Files

These files are in the directory **src/enc/software/ecc**. **These files should only be modified by HCC**.

File	Description
ecc.c	Common source code.
ecc.h	Header file for common code.
ecc_curve.c	Curve variables and functions.
ecc_nist.c	NIST source code.
ecc_nist.h	NIST header file.
ecdh.c	ECDH source code.
ecdsa.c	ECDSA source code.

2.4 Test Files

These files are in the directory **src/enc/test**. **These files should only be modified by HCC**.

File	Description
test_ecdh.c	ECDH test source code.
test_ecdsa.c	ECDSA test source code.

2.5 Version File

The file **src/version/ver_enc_sw_ecc.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_ecc.h**. This section lists the available configuration options and their default values.

ECDSA_INSTANCE_NR

The maximum number of ECDSA instances. The default is 1.

ECDH_INSTANCE_NR

The maximum number of ECDH instances. The default is 1.

ECC_MAX_NUMBER_LEN

The maximum size of the ECC point coordinate in bytes. The default is 68.

ECDH_TEST_ENABLE

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

ECDSA_TEST_ENABLE

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

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

ECC_TEST_ECDH_INITFN

The ECDH encryption driver initialization function. The default is *&ecdh_init_fn*.

ECC_TEST_ECDSA_INITFN

The ECDSA encryption driver initialization function. The default is *&ecdsa_init_fn*.

4 Application Programming Interface

This section describes the Application Programming Interface (API) functions, the ECDSA signature OIDs, and the error codes.

4.1 Functions

The functions are the following:

Function	Description
ecc_init()	Initializes the module and allocates the required resources.
ecc_start()	Starts the module.
ecc_stop()	Stops the module.
ecc_delete()	Deletes the module and releases the resources it used.
ecdh_init_fn()	Called from the EEM, registers the ECDH algorithm with it.
ecdsa_init_fn()	Called from the EEM, register the ECDSA algorithm with it.
ecdh_register_tests()	Registers the ECDH tests with the EEM test module.
ecdsa_register_tests()	Registers the ECDSA tests with the EEM test module.

ecc_init

Use this function to initialize the ECC module and obtain the required resources.

> **Note:** Call this before any other ECC function.

Format

```
t_ecc_ret ecc_init ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
ECC_SUCCESS	Successful execution.
ECC_ERROR	Operation failed; failed to obtain mutex.

ecc_start

Use this function to start the ECC module.

> **Note:** You must call **ecc_init()** before you call this function.

Format

```
t_ecc_ret ecc_start ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
ECC_SUCCESS	Successful execution.
ECC_ERROR	Operation failed.

ecc_stop

Use this function to stop the ECC module.

Format

```
t_ecc_ret ecc_stop ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
ECC_SUCCESS	Successful execution.
ECC_ERROR	Operation failed.

ecc_delete

Use this function to delete the ECC module, releasing the associated resources.

Format

```
t_ecc_ret ecc_delete ( void )
```

Arguments

Arguments
None.

Return Values

Return value	Description
ECC_SUCCESS	Successful execution.
ECC_ERROR	Operation failed; failed to delete mutex.

ecdh_init_fn

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

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

Format

```
t_enc_ret ecdh_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 ECDH functions.	t_enc_driver_fn * *

Return Values

Return value	Description
ECC_SUCCESS	Successful execution.
ECC_ERROR	The module has already been initialized.

ecdsa_init_fn

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

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

Format

```
t_enc_ret ecdsa_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 ECDSA functions.	t_enc_driver_fn * *

Return Values

Return value	Description
ECC_SUCCESS	Successful execution.
ECC_ERROR	The module has already been initialized.

ecdh_register_tests

Call this function to register the ECDH 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 ECDH_TEST_ENABLE configuration option must be set to 1 to enable this function.

Format

```
t_enc_ret ecdh_register_tests ( void )
```

Arguments

None.

Return Values

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

ecdsa_register_tests

Call this function to register the ECDSA 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 ECDSA_TEST_ENABLE configuration option must be set to 1 to enable this function.

Format

```
t_enc_ret ecdsa_register_tests ( void )
```

Arguments

None.

Return Values

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

4.2 OIDs

The ECDSA signature OIDs are defined in the file **src/api/api_enc_sw_ecc.h**.

Name	Value	Description
ECDSA_SHA1_ALG_OID	{ 0x2AU, 0x86U, 0x48U, 0xCEU, 0x3DU, 0x04U, 0x01U }	ECDSA with SHA-1: 1.2.840.10045.4.1.
ECDSA_SHA256_ALG_OID	{ 0x2AU, 0x86U, 0x48U, 0xCEU, 0x3DU, 0x04U, 0x03U, 0x02 }	ECDSA with SHA-256: 1.2.840.10045.4.3.2.
ECDSA_SHA384_ALG_OID	{ 0x2AU, 0x86U, 0x48U, 0xCEU, 0x3DU, 0x04U, 0x03U, 0x03 }	ECDSA with SHA-384: 1.2.840.10045.4.3.3.
ECDSA_SHA512_ALG_OID	{ 0x2AU, 0x86U, 0x48U, 0xCEU, 0x3DU, 0x04U, 0x03U, 0x04 }	ECDSA with SHA-512: 1.2.840.10045.4.3.4.
ECDSA_PUBLICKEY_OID	{ 0x2AU, 0x86U, 0x48U, 0xCEU, 0x3DU, 0x02U, 0x01U }	ECDSA public key OID.

4.3 Error Codes

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

Error code	Value	Meaning
ECC_SUCCESS	0	Successful execution.
ECC_ERROR	1	Operation failed.

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 system uses the following OAL components:

OAL Resource	Number Required
Tasks	0
Mutexes	1 per algorithm
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_memcmp()	psp_base	psp_string	Compares two blocks of memory.
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 files **psp/include/psp_endianness.h** and **psp/include/psp_array.h**.

Macro	Package	Element	Description
PSP_RD_BE16	psp_base	psp_endianness	Reads a 16 bit value stored as big-endian from a memory location.
PSP_WR_BE16	psp_base	psp_endianness	Writes a 16 bit value stored as big-endian to a memory location.
PSP_RD_8BITARRAY_OFFSET	psp_base	psp_array	Reads the offset in an 8 bit array.

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