# Ephemeral Diffie-Hellman Algorithm User Guide

Version 1.50 BETA

For use with Ephemeral Diffie-Hellman (EDH) Algorithm module versions 1.08 and above

**Date:** 22-Feb-2018 10:26

# 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 implement encryption using the Ephemeral Diffie-Hellman (EDH) algorithm.

## Overview

EDH is a key exchange mechanism for secure exchange of key material using asymmetric encryption. The goal of the protocol is to generate a secret value that is known only by the host and peer. This secret value cannot be computed by capturing publicly exchanged data.

Each execution of the encrypt or decrypt function completes based on the parameters passed to it, so normally there is no need for more than one instance of this driver. The only exception to this is if more than one application uses this and you want to ensure that a lower priority instance does not block a higher priority instance.

EDH uses the big number library for modulus calculations.

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

> **Note:** EDH is not popular now because it is computation-intensive and therefore can be slow.

### The Algorithm

The EDH algorithm uses modular calculation to provide asymmetric security. The peer and host generate their own secret values **a** and **b**.

The secret should be same size as the modulus (typically 128 or 256 bytes) - and should be randomly generated with a non-zero leading byte.

Each side uses the same key which consists of:

- **p** - modulus (must be a prime value).
- **g** - generator value.

HCC does not supply key generation software (there is plenty of specialist software available) and these keys are normally supplied by certificate authorities. The keys do not need to be secret. For example, many use the keys provided by the Oakley group. This is specified in RFC 2412, *The OAKLEY Key Determination Protocol*.

Each side calculates the public data and sends it to the peer.

- peer: **pp = $g^a$ mod p**
- host: **hp = $g^b$ mod p**

Peer and host now calculate the secret value by:

- **sec = $hp^a$ mod p = $pp^b$ mod p**

which can be shown to be equal because:

- **sec = $g^{(b*a)}$ mod p = $g^{(a*b)}$ mod p**

This secret value is then used to generate any other keys or secret values.

This algorithm implements the **$g^a$ mod p** calculation.

In EDH the secret value is generated at the start of each conversation. This method provides "future security" - because there is no way to determine the secret values in the future.

DH, the non-ephemeral version of the algorithm, relies on a system having a certificate which contains a public key and a private key; the secret is taken from the private key. DH is rarely used because it does not provide future security, meaning that if the private key is discovered later then older conversations can be decoded.

### Usage

Host and peer usage is symmetrical.

The host has three values:

- **ha** - host-generated secret value.
- **Key: p** - modulus and **g** - generator.
- Peer public data: **pp**

The peer has three values:

- **pa** - peer-generated secret value.
- **Key: p** - modulus and **g** - generator.
- Host public data: **hp**

### enc_driver_encrypt()

The EEM function **enc_driver_encrypt()** is used to calculate the passed public value. *p_in[]* points to the secret value generated by the host or peer (**ha** or **pa**).

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 key modulus (**p**) and generator value (**g**), each preceded by a two byte length field:<br><br><2 byte length of **p**><prime value **p**><2 byte length of **g**>< **g** value><br><br>All the values are formatted in big-endian mode. |
| ecd_key_size | uint16_t | The total length of the length fields + big number values' lengths. |

Other fields are discarded but should be set to NULL values on initialization.

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

The length of the output buffer must be greater than or equal to the length of the **p** modulus used in the key.

### enc_driver_decrypt()

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

*p_in[]* points to public data passed by the peer (**pp**). 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 buffer storing the key modulus (**p**) and host or peer secret value (**ha** or **pa**), each preceded by a two byte length field:<br><br><2 byte length of **p**><prime value **p**><2 byte length of **ha** or **pa**>< **ha** or **pa** value><br><br>All the values are formatted in big-endian mode. |
| ecd_key_size | uint16_t | The total length of the length fields + big number values' lengths. |

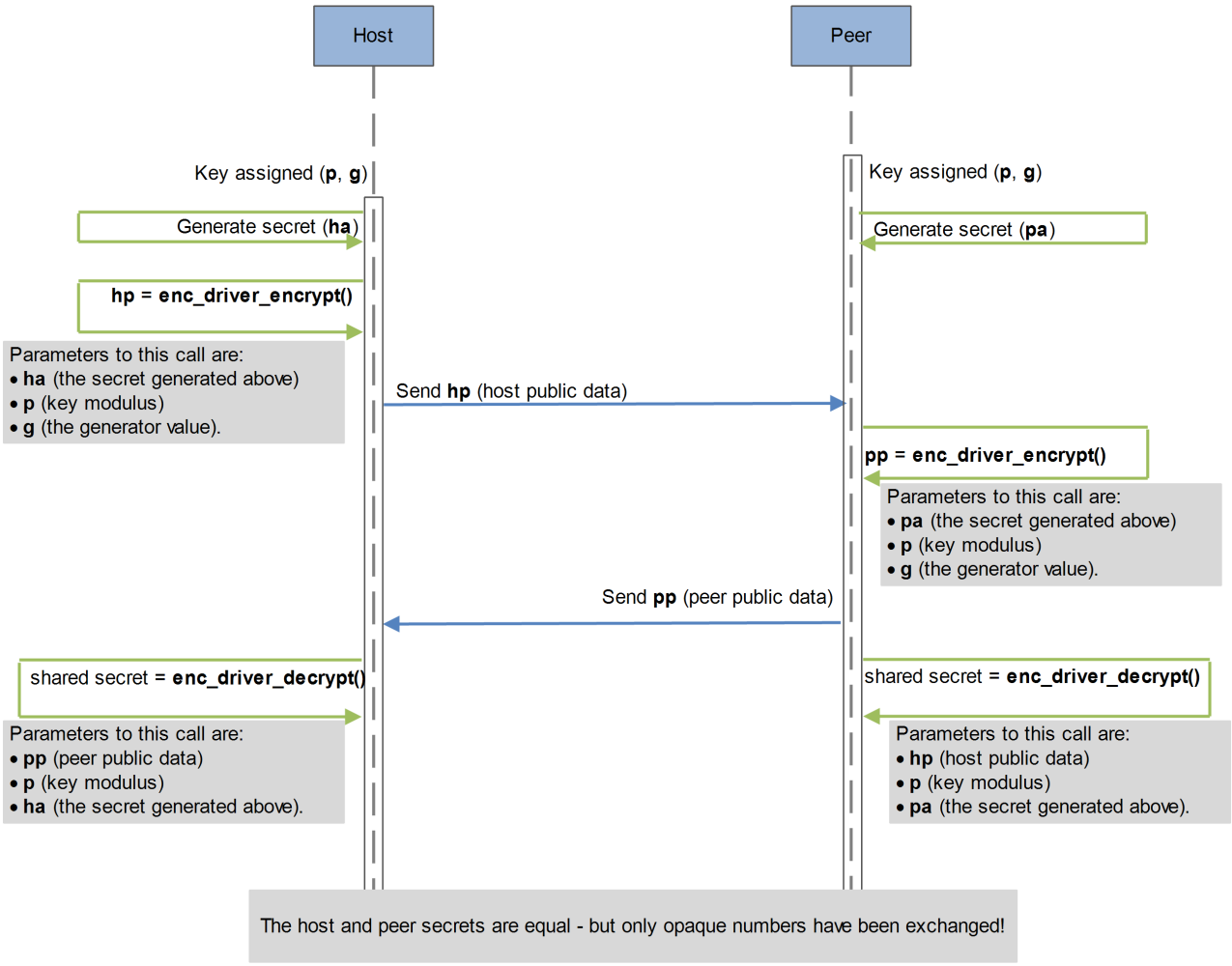
Other fields are discarded but should be set to NULL values on initialization.

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

The length of the output buffer must be greater than or equal to the length of the **p** modulus used in the key.

## 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.

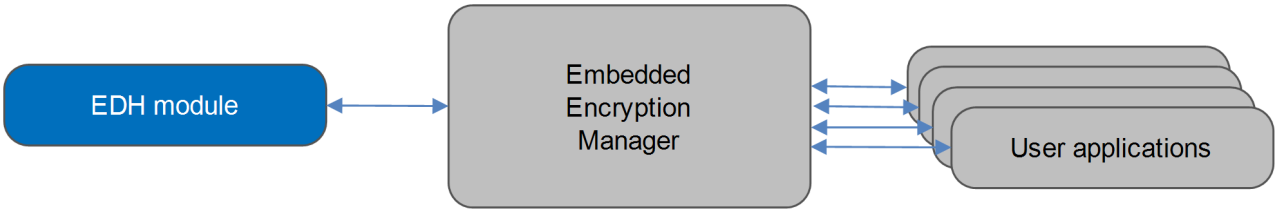
To implement a TRNG, a special hardware module is needed. Most modern chips implement a special module that can be used as a TRNG. If a device does not have a dedicated RND module, it can add an external random generator chip or implement the NIST-recommended random generator which uses RealTimeClock: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

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

## Using the Module

You register the EDH 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 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 EDH 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.
- 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_edh** | The EDH 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 base package.

**HCC Encryption Test Suite User Guide**

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

**HCC Ephemeral Diffie-Hellman Algorithm 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_edh.

The current version of this manual is 1.50 BETA. The full list of versions is as follows:

| Manual version | Date | Software version | Reason for change |
|---|---|---|---|
| 1.50B | 2018-02-22 | 1.08 | Extended *Introduction*, added new test options and function. |
| 1.40B | 2017-08-18 | 1.07 R2 | Updated *Packages* list. |
| 1.30B | 2017-06-15 | 1.07 R1 | New *Change History* format. |
| 1.20B | 2017-01-10 | 1.06 | Added lists of functions to API headers. |
| 1.10B | 2016-03-18 | 1.05 | Changed *Feature Check*. |
| 1.00B | 2015-02-11 | 1.04 | 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_edh.h** is the only file that should be included by an application using this module. It defines the edh_init_fn() function.

## 2.2 Configuration File

The file **src/config/config_enc_sw_edh.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/edh/edh.c** contains the EDH source code. **This file should only be modified by HCC**.

## 2.4 Test File

The file **src/enc/test/test_edh.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_edh.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_edh.h**. This section lists the available configuration options and their default values.

**EDH_INSTANCE_NR**

The maximum number of EDH instances. The default is 2.

Each execution of the **enc_driver_encrypt()** or **enc_driver_decrypt()** function completes based on the parameters passed to it, so normally there is no need for more than one instance of this driver. The only exception to this is if more than one application uses this and you want to ensure that a lower priority instance does not block a higher priority instance.

**EDH_BUF_LEN**

The EDH buffer size (the maximum supported size of a value used in EDH calculations). The length of input parameters or keys (the modulus or x value) cannot exceed this value. The default is 128.

**EDH_TEST_ENABLE**

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

**EDH_TEST_EDH_INITFN**

The EDH encryption driver initialization function. The default is *&edh_init_fn*; redefine this if you want to use another driver for a compatibility check.

# 4 Application Programming Interface

This section describes the Application Programming Interface (API) functions and the error codes.

## 4.1 Functions

Call the initialization function from the EEM to register the algorithm with it. Call the test function to register the EDH tests with the EEM test module.

The functions are the following:

| Function | Description |
|---|---|
| **edh_init_fn()** | Called from the EEM, this registers the EDH algorithm with it. |
| **edh_register_tests()** | Registers the EDH tests with the EEM test module. |

## edh_init_fn

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

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

### Format

```
t_enc_ret edh_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 EDH functions. | t_enc_driver_fn * * |

### Return Values

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

## edh_register_tests

Call this function to register the EDH 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 EDH_TEST_ENABLE configuration option must be set to 1 to enable this function.

### Format

```
t_enc_ret edh_register_tests ( void )
```

### Arguments

None.

### Return Values

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

## 4.2 Error Codes

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

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

# 5 Integration

The 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 |
| **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 macro:

| Macro | Package | Element | Description |
|---|---|---|---|
| PSP_RD_BE16 | psp_base | psp_endianness | Reads a 16 bit value stored as big-endian from a memory location. |

The module uses the following big number arithmetic functions from the EEM's Big Number Arithmetic API. These are described in the the HCC Embedded Encryption Manager User Guide.

| Function | Description |
|---|---|
| **bn_assign_be_buf()** | Assigns a little-endian buffer to a big number, based on a big-endian buffer. |
| **bn_get_be_buf()** | Exports a big number to a big-endian buffer. |
| **bn_get_power_modulo()** | Calculates *p_a* raised to the power of *p_e*, modulo *p_m*, and stores the result in *p_r*. |

**Note:** To improve performance, you can replace these functions with optimized or hardware-supported versions.