Advanced Encryption Standard User Guide

Version 1.60 BETA

For use with Advanced Encryption Standard (AES) module versions 1.18 and above

Date: 18-Sep-2017 15:08

Table of Contents

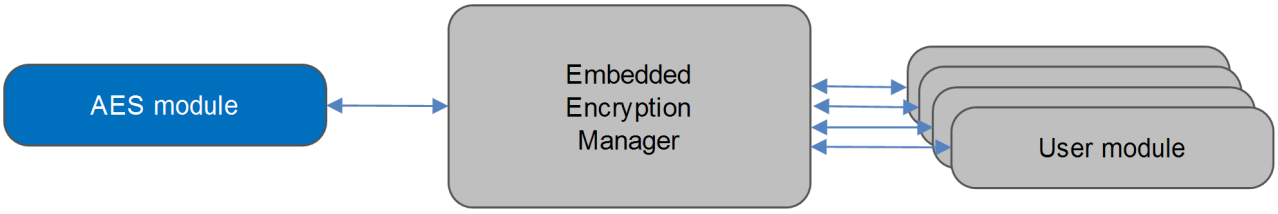
1 System Overview

1.1 Introduction

This guide is for those who want to implement bulk encryption using the Advanced Encryption Standard (AES). The AES uses a symmetric key algorithm, with the same key used to both encrypt and decrypt the data. The AES module implements the AES bulk encryption algorithm with Cipher Block Chaining (CBC). It supports AES RAW, AES CCM, AES CCM-8, AES CFB, AES CMAC, AES CTR, AES GCM, AES-XCBC-MAC, and AES-XCBC-MAC-96.

You register the AES 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:



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.

This module supports the following AES variations:

- AES RAW – no padding is added to the input data.
- AES CCM – Counter with CBC-MAC (CCM) mode. This is derived from the CTR mode (see below).
- AES CFB – Cipher Feedback (CFB) mode. This turns a block cipher into a self-synchronizing stream cipher.
- AES CMAC – AES Cipher-based Message Authentication Code (CMAC).
- AES CTR – the counter (CTR) mode of AES. Again no padding is added to the input data. This is compatible with Encapsulating Security Payload (ESP), one of the Internet Protocol Security (IPsec) protocols.
- AES GCM – Galois/Counter Mode (GCM): encrypted data output is composed of cipher text + TAG (16bytes).

- AES-XCBC-MAC and AES-XCBC-MAC-96 – here a Message Authentication Code (MAC) value is calculated.

1.2 Feature Check

The main features of the AES 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 HCC's Embedded Encryption Manager (EEM) standard and is compatible with the EEM.
- Supports AES RAW and AES CTR (RFC 3686).
- Supports AES-CCM and AES-CCM-8 (RFC 3610).
- Supports AES CFB.
- Supports AES-CMAC and AES-CMAC-96 (RFC 4493).
- Supports AES GCM – Galois/Counter Mode (GCM), based on http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf
- Supports AES-XCBC-MAC and AES-XCBC-MAC-96 (RFC 3566).
- Can be verified by using the HCC Encryption Test Suite.

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_aes	The AES 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 Advanced Encryption Standard 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 Archive: Advanced Encryption Standard User Guide.
- For the history of changes made to the package code itself, see History: enc_aes.

The current version of this manual is 1.60 BETA. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.60B	2017-09-18	1.18	Added AES CCM and AES CCM 8.
1.50B	2017-06-15	1.17	Added AES GCM, macros for TLS. New *Change History* format.
1.40B	2017-05-05	1.16	Added AES CBC.
1.30B	2017-04-12	1.14	Added AES CFB, AES 192 keys.
1.20B	2017-01-10	1.10	Added lists of functions to API headers.
1.10B	2016-03-09	1.08	Added Change History.
1.00B	2015-02-11	1.06	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_aes.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_aes.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/aes/aes.c** contains the source code.

This file should only be modified by HCC.

2.4 Version File

The file **src/version/ver_enc_sw_aes.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_aes.h**.

AES_INSTANCE_NR

The maximum number of AES algorithm instances. The default is 1.

AES_RAW_INSTANCE_NR

The maximum number of AES RAW instances. The default is 1.

AES_CFB_INSTANCE_NR

The maximum number of AES CFB instances. The default is 1.

AES_CTR_INSTANCE_NR

The maximum number of AES CTR instances. The default is 1.

AES_XCBC_MAC_INSTANCE_NR

The maximum number of AES XCBC instances. The default is 1.

AES_CMAC_INSTANCE_NR

The maximum number of AES CMAC instances. The default is 1.

AES_GCM_INSTANCE_NR

The maximum number of AES GCM instances. The default is 1.

AES_CCM_INSTANCE_NR

The maximum number of AES CCM instances. The default is 1.

AES_TLS12_PADDING_METHOD

This controls padding generation. The values are:

- 0 (the default) – padding is generated consistent with PKCS #7 (RFC 5652, section 6.3).
- 1 – use this for TLS 1.2 encryption. It generates padding in a manner consistent with RFC 5246, section 6.2.3.2.

4 Application Programming Interface

This section describes the Application Programming Interface (API) functions, the key lengths, output buffer lengths, AES-CTR parameters and the error codes.

4.1 Functions

The functions are the following:

Function	Description
aes_init_fn()	Called from the EEM, forwards the structure containing AES functions to it.
aes_ccm_init_fn()	Called from the EEM, forwards the structure containing AES CCM functions to it.
aes_ccm_8_init_fn()	Called from the EEM, forwards the structure containing AES CCM-8 functions to it.
aes_cfb_init_fn()	Called from the EEM, forwards the structure containing AES CFB functions to it.
aes_cmac_init_fn()	Called from the EEM, forwards the structure containing AES CMAC functions to it.
aes_ctr_init_fn()	Called from the EEM, forwards the structure containing AES CTR functions to it.
aes_gcm_init_fn()	Called from the EEM, forwards the structure containing AES GCM functions to it.
aes_raw_init_fn()	Called from the EEM, forwards the structure containing AES RAW functions to it.
aes_xcbc_mac_init_fn()	Called from the EEM, forwards the structure containing AES AES-XCBC-MAC or AES-XCBC-MAC-96 functions to it.

aes_init_fn

Call this function from the EEM to forward the structure containing AES functions to it.

AES adds padding bytes to the input data. Specify the padding method by using the configuration option AES_TLS12_PADDING_METHOD.

Format

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

Arguments

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

Return Values

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

aes_ccm_init_fn

Call this function from the EEM to forward the structure containing Counter with CBC-MAC (CCM) functions to it. This initializes AES CCM, based on RFC 3610.

The initialization vector's size range is 7 - 13 bytes.

Additional authorization data can be left undefined.

The output buffer minimum sizes are:

- encryption data_len + 8U
- decryption data_len - 8U

Format

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

Arguments

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

Return Values

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

aes_ccm_8_init_fn

Call this function from the EEM to forward the structure containing Counter with CBC-MAC-8 (CCM-8) functions to it. This initializes AES CCM-8, based on RFC 3610.

The initialization vector's size range is 7 - 13 bytes.

Additional authorization data can be left undefined.

The output buffer minimum sizes are:

- encryption data_len + 8U
- decryption data_len - 8U

Format

```
t_enc_ret aes_ccm_8_init_fn ( t_enc_driver_fn const * * const pp_encdriver )
```

Arguments

Parameter	Description	Type
pp_encdriver	A pointer to a structure containing AES CCM-8 functions.	t_enc_driver_fn * *

Return Values

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

aes_cfb_init_fn

Call this function from the EEM to forward the structure containing AES CFB functions to it.

No padding is added to the input data. There is no restriction on the input data length of encryption /decryption functions.

Format

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

Arguments

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

Return Values

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

aes_cmac_init_fn

Call this function from the EEM to forward the structure containing AES CMAC (96) functions to it.

The output data is the MAC value.

> **Note:** To use AES-CMAC-96 mode, set the Output Buffer Length to AES_CMAC_96_OUT.

Format

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

Arguments

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

Return Values

Return value	Description
ENC_SUCCESS	Successful execution. The value returned is the MAC value.
ENC_INVALID_ERR	The module has already been initialized.

aes_ctr_init_fn

Call this function from the EEM to forward the structure containing AES CTR functions to it.

This is the counter (CTR) version of the AES algorithm. This includes ESP compatibility, so it it accepts an eight byte IV vector. The NONCE and counter value is passed with the key.

Format

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

Arguments

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

Return Values

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

aes_gcm_init_fn

Call this function from the EEM to forward the structure containing GCM functions to it.

This initializes AES-GCM based on http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf

Encrypted data output is composed of cipher text + TAG(16bytes).

Format

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

Arguments

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

Return Values

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

aes_raw_init_fn

Call this function from the EEM to forward the structure containing RAW AES functions to it.

RAW AES means that no padding is added to the input data. The data length for encryption and decryption must be a multiple of the AES block size (16).

Format

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

Arguments

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

Return Values

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

aes_xcbc_mac_init_fn

Call this function from the EEM to forward the structure containing AES AES-XCBC-MAC or AES-XCBC-MAC-96 functions to it. This initializes AES-XCBC-MAC (96), based on RFC 3566.

The output data is the MAC value.

> **Note:** To use AES-XCBC-MAC-96 mode, set the Output Buffer Length to AES_XCBC_MAC_96_OUT.

Format

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

Arguments

Parameter	Description	Type
pp_encdriver	A pointer to a structure containing AES AES-XCBC-MAC or AES-XCBC-MAC-96 functions.	t_enc_driver_fn * *

Return Values

Return value	Description
ENC_SUCCESS	Successful execution. The returned value is the MAC value.
ENC_INVALID_ERR	The module has already been initialized.

4.2 Types and Definitions

AES-CTR Parameters

Set these parameters for operating AES-CTR in compatibility with ESP.

Name	Value	Description
AES_CTR_ESP_IV_SIZE	8	Initialization vector.
AES_CTR_ESP_128_KEY_SIZE	24	128 AES key size + 8 bytes of hidden Initialization vector.
AES_CTR_ESP_256_KEY_SIZE	40	256 AES key size + 8 bytes of hidden Initialization vector.

fixed_iv_length values

The following values control the *fixed_iv_length* parameter in TLS for CCM and GCM:

Name	Value	Description
AES_CCM_TLS_FIXED_IV_LENGTH	4	Value of *fixed_iv_length* parameter in TLS for CCM.
AES_CCM_TLS_RECORD_IV_LENGTH	8	Value of *record_iv_length* parameter in TLS for CCM (for configuring IV in cipher suite).
AES_GCM_TLS_FIXED_IV_LENGTH	4	Value of *fixed_iv_length* parameter in TLS for GCM.
AES_GCM_TLS_RECORD_IV_LENGTH	8	Value of *record_iv_length* parameter in TLS for GCM (for configuring IV in cipher suite).

Key Lengths

The key lengths are as follows:

Name	Value	Description
AES_128_KEY_LEN	16	128 bit AES key length in bytes.
AES_192_KEY_LEN	24	192 bit AES key length in bytes.
AES_256_KEY_LEN	32	256 bit AES key length in bytes.

Output Buffer Lengths

Set the encryption output buffer length to the relevant value below:

Name	Value	Description
AES_CCM_OUT	16	Output size for AES-CCM.
AES_CCM_8_OUT	8	Output size for AES-CCM-8.
AES_CMAC_OUT	16	Output size for AES-CMAC.
AES_CMAC_96_OUT	12	Output size for AES-CMAC-96.
AES_XCBC_MAC_OUT	16	Output size for AES-XCBC-MAC.
AES_XCBC_MAC_96_OUT	12	Output size for AES-XCBC-MAC-96.

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

This section describes all aspects of the module that require integration with your target project. This includes porting and configuration of external resources.

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 function:

Function	Package	Element	Description
psp_memcmp()	psp_base	psp_string	Compares two blocks of memory.
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_BE32	psp_base	psp_endianness	Reads a 32 bit value stored as big-endian from a memory location.
PSP_RD_LE32	psp_base	psp_endianness	Reads a 32 bit value stored as little-endian from a memory location.
PSP_WR_BE32	psp_base	psp_endianness	Writes a 32 bit value to be stored as big-endian to 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.