Triple Data Encryption Standard User Guide

Version 1.30

For use with Triple Data Encryption Standard (3DES) module versions 1.09 and above

Date: 22-Feb-2018 10:44

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 bulk encryption using the Data Encryption Standard (DES) or Triple Data Encryption Standard (3DES). The 3DES uses a symmetric key algorithm, with the same key used both to encrypt and decrypt the data. The 3DES module implements the Triple Data Encryption Standard bulk encryption algorithm.

Note:

- 3DES is rarely used now. Where it is used, this is always in block cipher mode which is the only mode HCC supports.
- AES has replaced 3DES for most use cases. For details of HCC's AES module, see the HCC Advanced Encryption Standard (AES) User Guide.

Overview

HCC's DES and 3DES use Cipher Block Chaining (CBC) mode. This module provides two types of 3DES encryption/decryption, as follows.

Raw DES

This is the original form of DES and uses a single DES iteration. This is used by the Network Time Protocol (NTP) for authorization.

Raw DES can only encrypt/decrypt data aligned to 8 bytes at a time. When not raw, DES can process any length of data. The algorithm then provides padding.

Triple DES (3DES)

This superseded DES and provides improved security. It is available with and without padding.

The 3DES encryption algorithm calculates DES three times so gets its name triple DES. It uses a 24 byte key divided into 3 parts. It operates over three iterations as follows:

1. To encrypt, first encrypt with the first 8 bytes of the key.
2. Decrypt with the second 8 bytes of the key
3. Encrypt with the last 8 bytes.

Decryption is the reverse process.

The implementation is not stateful.

The following modes are not supported:

- Cipher feedback.
- Electronic code book.

- Feedback output.
- Counter mode.

TDES with Auto-padding

This driver automatically adds padding to the encrypted data so the input data does not have to be multiple of 8 bytes. The driver works in CBC mode.

Note: The initialization vector does not need to be secret but it must be different for different runs of the algorithm and has to be known to both the writer and reader of the data. Its purpose is to ensure that if similar data is encrypted it will be encoded differently; for example if two files both start with the same header information then the encrypted code would be identical and could provide an attack vector for someone attempting to decrypt the data.

enc_driver_encrypt()

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

p_in[] points to the data to be encrypted. 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_init_vect	uint8_t *	A pointer to the 8 byte initialization vector; see the note above.
ecd_init_vect_size	uint16_t	The length of the initial data vector, always 8.
p_ecd_key	void *	A pointer to the buffer storing the 3DES key.
ecd_key_size	uint16_t	The length of the key, 24 bytes.

Other fields are discarded but should be set to NULL.

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

The output length, *p_out_len*, must be set to the output buffer size. This must be a multiple of 8 bytes and greater than the input data length.

enc_driver_decrypt()

The EEM function **enc_driver_decrypt()** is used to decrypt input data.

p_in[] points to the data to be decrypted. The length of the data (*in_len*) must be a multiple of 8 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 8 byte initialization vector; see the note above.
ecd_init_vect_size	uint16_t	The length of the initial data vector, always 8.
p_ecd_key	void *	A pointer to the buffer storing the 3DES key.
ecd_key_size	uint16_t	The length of the key, 24 bytes.

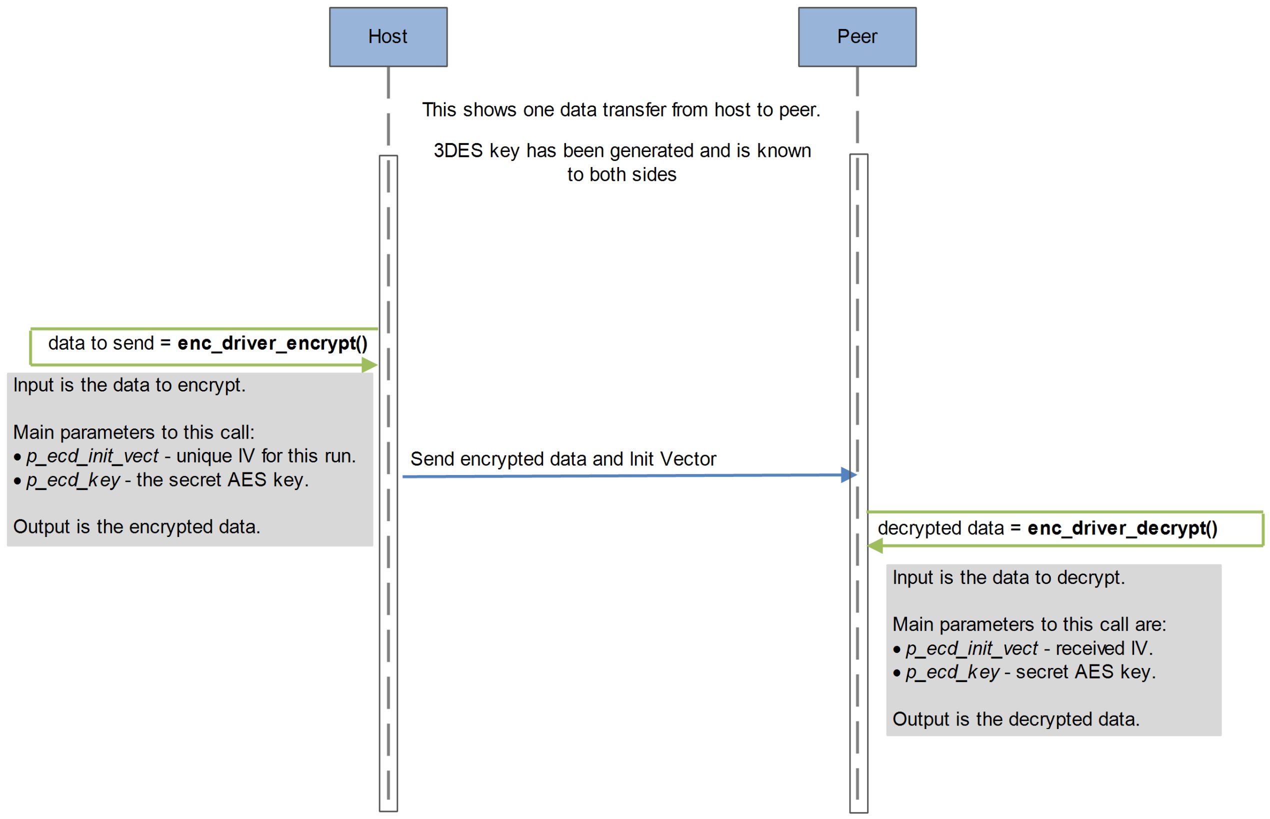
Other fields are discarded but should be set to NULL.

The output data from **enc_driver_decrypt()** is the decrypted data, *p_out[]*.

The output length, *p_out_len*, must be set to the output buffer size. This must be greater than or equal to the input data length.

Sequence Diagram

The following sequence diagram shows the process:


Host
Peer
This shows one data transfer from host to peer.
3DES key has been generated and is known to both sides
data to send = **enc_driver_encrypt()**
Input is the data to encrypt.
Main parameters to this call:
• *p_ecd_init_vect* - unique IV for this run.
• *p_ecd_key* - the secret AES key.
Output is the encrypted data.
Send encrypted data and Init Vector
decrypted data = **enc_driver_decrypt()**
Input is the data to decrypt.
Main parameters to this call are:
• *p_ecd_init_vect* - received IV.
• *p_ecd_key* - secret AES key.
Output is the decrypted data.

TDES Raw (with no padding)

This driver does not add padding to the encrypted data so the input data have to be a multiple of 8 bytes. The driver works in CBC mode.

Note: The initialization vector does not need to be secret but it must be different for different runs of the algorithm and has to be known to both the writer and reader of the data. Its purpose is to ensure that if similar data is encrypted it will be encoded differently; for example if two files both start with the same header information then the encrypted code would be identical and could provide an attack vector for someone attempting to decrypt the data.

enc_driver_encrypt()

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

p_in[] points to the data to be encrypted. The length of the data (*in_len*) must be a multiple of 8 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 8 byte initialization vector; see the note above.
ecd_init_vect_size	uint16_t	The length of the initial data vector, always 8.
p_ecd_key	void *	A pointer to the buffer storing the 3DES key.
ecd_key_size	uint16_t	The length of the key, 24 bytes.

Other fields are discarded but should be set to NULL.

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

The output length, *p_out_len*, must be set to the output buffer size. This must be a multiple of 8 bytes and greater than the input data length.

enc_driver_decrypt()

The EEM function **enc_driver_decrypt()** is used to decrypt input data.

p_in[] points to the data to be decrypted. The length of the data (*in_len*) must be a multiple of 8 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 8 byte initialization vector; see the note above.
ecd_init_vect_size	uint16_t	The length of the initial data vector, always 8.
p_ecd_key	void *	A pointer to the buffer storing the 3DES key.
ecd_key_size	uint16_t	The length of the key, 24 bytes.

Other fields are discarded but should be set to NULL.

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

The output length, *p_out_len*, must be set to the output buffer size. This must be greater than or equal to the input data length.

DES RAW

This driver does not add padding to the encrypted data so the input data have to be a multiple of 8 bytes. The driver works in CBC mode.

Note: The initialization vector does not need to be secret but it must be different for different runs of the algorithm and has to be known to both the writer and reader of the data. Its purpose is to ensure that if similar data is encrypted it will be encoded differently; for example if two files both start with the same header information then the encrypted code would be identical and could provide an attack vector for someone attempting to decrypt the data.

enc_driver_encrypt()

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

p_in[] points to the data to be encrypted. The length of the data (*in_len*) must be a multiple of 8 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 8 byte initialization vector; see the note above.
ecd_init_vect_size	uint16_t	The length of the initial data vector, always 8.
p_ecd_key	void *	A pointer to the buffer storing the DES key.
ecd_key_size	uint16_t	The length of the key, 8 bytes.

Other fields are discarded but should be set to NULL.

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

The output length, *p_out_len*, must be set to the output buffer size. This must be a multiple of 8 bytes and greater than the input data length.

enc_driver_decrypt()

The EEM function **enc_driver_decrypt()** is used to decrypt input data.

p_in[] points to the data to be decrypted. The length of the data (*in_len*) must be a multiple of 8 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 8 byte initialization vector; see the note above.
ecd_init_vect_size	uint16_t	The length of the initial data vector, always 8.
p_ecd_key	void *	A pointer to the buffer storing the DES key.
ecd_key_size	uint16_t	The length of the key, 8 bytes.

Other fields are discarded but should be set to NULL.

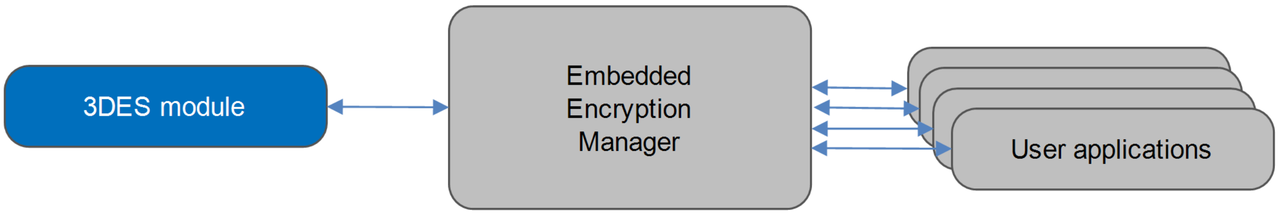
The output data from **enc_driver_decrypt()** is the decrypted data, stored in *p_out[]*.

The output length, *p_out_len*, must be set to the output buffer size. This must be greater than or equal to the input data length.

Using the Module

You register the 3DES 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 3DES 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.
- Provides Triple DES and raw DES options.
- 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_tdes	The 3DES 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 Triple Data Encryption Standard 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_tdes.

The current version of this manual is 1.30 BETA. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.30 BETA	2018-02-22	1.09	Extended *Introduction*, added new test options and function.
1.20	2017-06-15	1.08	New *Change History* format.
1.10	2016-03-11	1.07	Added TDES Raw.
1.10 BETA	2016-01-29	1.05	Added *Change History*.
1.00	2015-02-11	1.05	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_tdes.h** is the only file that should be included by an application using this module. It defines the Application Programming Interface.

2.2 Configuration File

The file **src/config/config_enc_sw_tdes.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/tdes/tdes.c** is the source code file. **This file should only be modified by HCC**.

2.4 Test File

The file **src/enc/test/test_tdes.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_tdes.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 Option

Set the system configuration options in the file **src/config/config_enc_sw_tdes.h**. This section lists the available configuration options and their default values.

TDES_TLS12_PADDING_METHOD

This controls padding generation. The values are:

- 0 (the default) – padding is generated consistent with PKCS #7. For details, see 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.

TDES_TEST_ENABLE

Keep the default of 1 to enable the 3DES test suite. Otherwise, set this to 0.

The following options set the 3DES tests' init functions; redefine these if you want to use another set of drivers for a compatibility check.

TDES_TEST_TDES_INITFN

The 3DES encryption driver init function. The default is &tdes_init_fn.

TDES_TEST_TDES_RAW_INITFN

The 3DES RAW encryption driver init function. The default is &tdes_raw_init_fn.

TDES_TEST_DES_RAW_INITFN

The DES RAW encryption driver init function. The default is &des_raw_init_fn.

4 Application Programming Interface

This section describes the Application Programming Interface (API) functions, the key length, 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 TDES tests with the EEM test module.

The functions are the following.

Function	Description
tdes_init_fn()	Called from the EEM, registers 3DES encryption/decryption with it. This forwards the structure containing 3DES functions to the EEM.
des_raw_init_fn()	Called from the EEM, registers RAW DES encryption/decryption wit h it . This forwards the structure containing RAW DES functions to the EEM.
tdes_raw_init_fn()	Called from the EEM, registers RAW 3DES encryption/decryption w ith it . This forwards the structure containing RAW 3DES functions to the EEM.
tdes_register_tests()	Registers the TDES tests with the EEM test module.

tdes_init_fn

Call this function once from the EEM to register 3DES encryption/decryption with it.

This forwards the *t_enc_driver_fn* structure containing 3DES functions to the EEM. This structure is described in the the *HCC Embedded Encryption Manager User Guide*.

Format

```
t_enc_ret tdes_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 3DES functions.	t_enc_driver_fn * *

Return Values

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

des_raw_init_fn

Call this function once from the EEM to register RAW DES encryption/decryption.

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

In RAW mode no padding is added. The size of the data to be encrypted must be a multiple of 8.

Format

```
t_enc_ret des_raw_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 RAW DES functions.	t_enc_driver_fn * *

Return Values

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

tdes_raw_init_fn

Call this function once from the EEM to register RAW 3DES encryption/decryption with it.

This forwards the *t_enc_driver_fn* structure containing RAW 3DES functions to the EEM. This structure is described in the the *HCC Embedded Encryption Manager User Guide*.

In RAW mode no padding is added. The size of the data to be encrypted must be a multiple of 8.

Format

```
t_enc_ret tdes_raw_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 RAW 3DES functions.	t_enc_driver_fn * *

Return Values

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

tdes_register_tests

Call this function to register the TDES 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 TDES_TEST_ENABLE configuration option must be set to 1 to enable this function.

Format

```
t_enc_ret tdes_register_tests ( void )
```

Arguments

None.

Return Values

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

4.2 Key Length

The key length is defined in the file **src/api/api_enc_sw_tdes.h**.

Name	Value	Description
TDES_INIVECT_SIZE	8	The 3DES initialization vector size.
TDES_KEY_LEN	24	The key length in bytes.

4.3 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

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 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.