Secure Hash Algorithm (SHA) User Guide

Version 1.20 BETA

For use with SHA module versions 1.7 and above

Date: 11-Jan-2017 13:55

Table of Contents

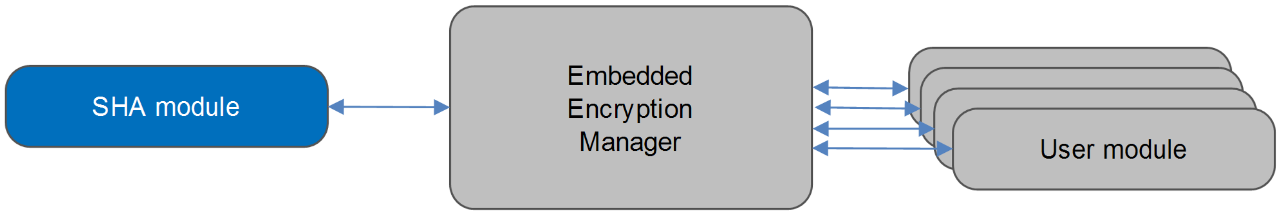
1 System Overview

1.1 Introduction

This guide is for those who want to generate hash codes for data by using any of the following Secure Hash Algorithms (SHAs): SHA-1, SHA-1 HMAC, SHA1-HMAC-96, SHA-256, SHA-384 or SHA-512. (HMAC stands for Hash Message Authentication Code.) The SHA module implements all these hash algorithms.

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:



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.
- Can be verified by using the HCC Encryption Test Suite.
- The SHA-1 implementation conforms to RFC 3174.
- The SHA-256, SHA-384, and SHA-512 implementations conform to RFC 4634.

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 Secure Hash Algorithm (SHA) User Guide

This is this document.

1.4 Change History

This section includes recent changes to this product. For a list of all changes, refer to the file **hcc/history/enc/enc_sha.txt** in the distribution package.

Version	Changes
1.07	SHA1-HMAC and SHA1-HMAC-96 are now stateful. The result digest is returned when output buffer is not NULL.
1.06	Added stateful implementation of SHA1, SHA256, SHA512 and SHA384. The digest is calculated when the output buffer of *enc_driver_hash* is not NULL, otherwise the result is accumulated.
1.05	Added SHA OID numbers to the API.
1.04	Added support for SHA1-HMAC and SHA1-HMAC-96. The new functions are **sha1_hmac_init_fn()** and **sha1_hmac_96_init_fn()**
1.03	Added support for SHA-512 and SHA-384. Corrected output length check in SHA-256.
1.02	Added SHA1_BLOCK_LEN, SHA256_BLOCK_LEN defines to API. Simplified SHA-1 and SHA-256 calculation algorithm.
1.01	Changed version number to match new **enc_base** package.

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** contains the source code. **This file should only be modified by HCC**
.

2.4 Platform Support Package (PSP) File

The file **psp/include/psp_endianness.h** provides macros for reading and writing little- and big-Endian values.

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

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 configuration 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.

4 Application Programming Interface

This section describes the Application Programming Interface (API) functions, the hash output sizes, and the error codes.

4.1 Functions

The functions are the following. They are all called from the EEM.

Function	Description
sha1_init_fn()	This forwards the structure containing SHA-1 functions to the EEM.
sha1_hmac_init_fn()	This forwards the structure containing SHA1-HMAC functions to the EEM.
sha1_hmac_96_init_fn()	This forwards the structure containing SHA1-HMAC-96 functions to the EEM.
sha256_init_fn()	This forwards the structure containing SHA-256 functions to the EEM.
sha384_init_fn()	This forwards the structure containing SHA-384 functions to the EEM.
sha512_init_fn()	This forwards the structure containing SHA-512 functions to the EEM.

sha1_init_fn

Call this function from the EEM to forward the structure containing SHA-1 functions to it.

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 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 function from the EEM to forward the structure containing SHA1-HMAC functions to it.

The driver implements SHA1-HMAC (Hash Message Authentication Code). Use the driver as though it would encrypt data. The encryption function calculates an HMAC value. The HMAC output length is equal to SHA1_HMAC_OUT_LEN.

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 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 function from the EEM to forward the structure containing SHA1-HMAC-96 functions to it.

The driver implements SHA1-HMAC-96. Use the driver as though it would encrypt data. The encryption function calculates an HMAC (Hash Message Authentication Code) value. The HMAC output length is equal to SHA1_HMAC_96_OUT_LEN. The key length is restricted to 20 bytes (RFC2404).

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 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 function from the EEM to forward the structure containing SHA-256 functions to it.

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 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 function from the EEM to forward the structure containing SHA-384 functions to it.

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 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 function from the EEM to forward the structure containing SHA-512 functions to it.

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 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.

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	4
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.