RSA Signature Algorithm User's Guide

Version 1.00 BETA

For use with RSA versions 1.3 and above

Date: 11-Feb-2015 11:43

Table of Contents

Version 1.00 BETA

For use with RSA versions 1.3 and above

Encryption Documents Home

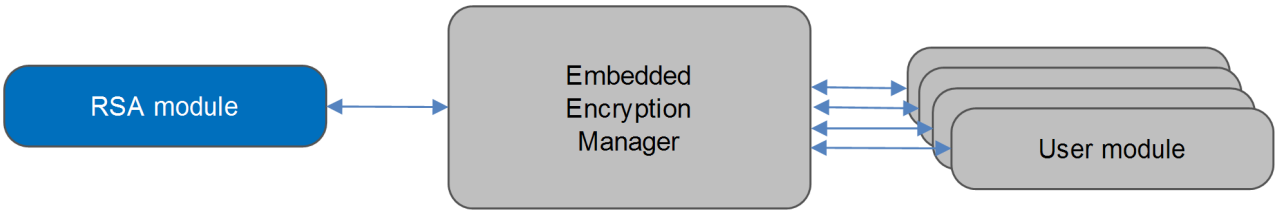
1 System Overview

1.1 Introduction

This guide is for those who want to implement encryption using the Rivest, Shamir and Adelman (RSA) signature algorithm. The RSA algorithm is most often used for public/private key encryption and decryption. Typically data is sent after it has been encrypted using the public key of the intended receiver, but can only be decrypted using the private key of the receiver.

You register the RSA module with HCC's Embedded Encryption Manager (EEM), making it usable by other applications (for example, HCC's TLS/SSL) 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 RSA module are the following:

- It conforms to the HCC Advanced Embedded Framework.
- It conforms to the HCC Coding Standard including full MISRA compliance.
- It conforms to the HCC Embedded Encryption Manager (EEM) standard and is compatible with the EEM.
- It 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_rsa	The RSA package described in this document.

Documents

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's Guide

This document describes the EEM.

HCC RSA Signature Algorithm User's Guide

This is this document.

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_rsa.h** is the only file that should be included by an application using this module. It defines the rsa_init_fn() function.

2.2 Configuration File

The file **src/config/config_enc_sw_rsa.h** contains the single configurable system parameter. Configure this as required. This is the only file in the module that you should modify.

2.3 System File

The file **src/enc/software/rsa/rsa.c** is the source code file. **This file should only be modified by HCC**.

2.4 Version File

The file **src/version/ver_enc_sw_rsa.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 single system configuration option in the file **src/config/config_enc_sw_rsa.h**.

RSA_INSTANCE_NR

The maximum number of RSA driver instances. The default is 2.

4 Application Programming Interface (API)

This section describes the single API function, the key lengths, and the error codes.

4.1 rsa_init_fn

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

Format

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

Arguments

Parameter	Description	Type
pp_encdriver	A pointer to a structure containing RSA 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 Key Lengths

The key lengths are defined in the file **src/api/api_enc_sw_rsa.h**.

Name	Value	Description
RSA_MIN_PUB_KEY_LEN	64U	The minimum public key length in bytes.
RSA_MAX_PUB_KEY_LEN	256U	The maximum public 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	0U	Successful execution.
ENC_INVALID_ERR	1U	The module has already been initialized.

5 Integration

The RSA 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 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 functions, see the *Platform Support Package (PSP) Base User's 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_WR_8BITARRAY_OFFSET()	psp_base	psp_array32	

The module makes use of the following big number arithmetic functions from the Embedded Encryption Manager's big number module.

Note: To improve performance, you can replace these functions with optimized or hardware-supported versions. For details, see the *HCC Embedded Encryption Manager User's Guide*.

Function	Description
bn_compare()	Compares two large numbers.
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*.