RSA™ BSAFE® Crypto-C

Toolkit Module Security Policy

Version 4.31
September 21, 2000

RSA Data Security, Inc.
2955 Campus Drive, Suite 400
San Mateo, California 94403-2507

Revision History

Table 1: Revision History

Revision	Date	Author	Description
4.11	2/8/99	David Young	Initial NIST Submission
4.11	2/22/99	David Young	Temporarily removes X9.31 from FIPS140 Mode. To be reinstated in Version 4.2
4.11	2/24/99	Eliza Sachs/ Dave Young	Merge in corrections from technical writing group.
4.11	4/05/99	Dave Young	Add support in policy for key distribution and import. Fix some naming inconsistencies.
4.11	4/15/99	Dave Young	Modify per NIST comments
4.11	4/16/99	Dave Young	Re-sequence, eliminate numbering system, add in edits from Eliza Sachs
4.11	4/21/99	Dave Young	Policy reflects that Module automatically erase secret key after startup verification
4.31	12/16/99	Dave Young	Update for Release 4.31
4.31	1/04/99	Dave Young	Update per Cygnacom Comments

Introduction

The RSA BSAFE Crypto-C Toolkit Module (Crypto-C Module) is a software module, implemented as a 32-bit Windows™ ‘NT compatible DLL, which provides a variety of cryptographic services that are accessed by calls from C-language programs through an Application Program Interface (API). This API is documented in the *RSA BSAFE Crypto-C Library Reference Manual*. The Module is accessed from your C-language programs using the same method as the BSAFE Crypto-C static toolkit, via the inclusion of the include file “BSAFE.h”. With a few exceptions, you can use the same API calls that are used in the BSAFE Crypto-C Toolkit Static Library. In addition, the Crypto-C Module enables OEM software developers to construct Crypto-C -based applications that rely on FIPS140-1 certified cryptographic functions.

The module supports the FIPS approved DSA, rDSA (RSA ANSI X9.31), DES and TDES Modes, and SHA-1 algorithms. It also provides non-FIPS approved RSA Encryption/Decryption, MD2, MD5, HMAC, DESX, RC2, RC4, RC5, Elliptic Curve (F_2&F_p), Elliptic Curve Encryption Scheme, Elliptic Curve DSA, and Bloom-Shamir algorithms.

Existing applications can easily be modified to take advantage of the BSAFE Crypto-C Module. To do so, just change the name of the algorithm chooser in your source code to SAFETY_ALGORITHM_CHOOSER (see the section *The Algorithm Chooser* in Chapter 1 of the *RSA BSAFE Crypto-C Library Reference Manual).*

Overview of FIPS 140-1 Categories

The RSA BSAFE Crypto-C Toolkit Module conforms to FIPS140-1 Level 1 as shown in *Table I, Category Levels tested for the RSA BSAFE Crypto-C Toolkit Module*.

Table I, Category Levels tested for the RSA BSAFE Crypto-C Toolkit Module

FIPS140-1 TEST CATEGORY	LEVEL
Cryptographic Modules	1
Module Interfaces	1
Roles & Services	1
Finite State Machine Model	1
Physical Security	1
Software Security	1
Operating System Security	1
Key Management	1
Cryptographic Algorithms	1
EMI/EMC	3[JH1]
Self-Tests	1

Overview of Features

Cryptographic Modules

The Crypto-C Module is viewed in FIPS140-1 terms as a “multi-chip standalone module.” To come up with a complete platform, RSA Data Security, Inc. provides the software and you add the hardware, an IBM Compatible PC. A “secure cryptographic boundary” is defined for FIPS140-1 purposes as those applicable software and hardware components internal to a host IBM-compatible PC that is running the Windows™ ‘NT Operating System (OS).

Additionally, the Windows™ ‘NT OS imposes rules that segregate user processes into protected memory spaces called “process spaces.” Each such process space belongs to a single user. Access to the process space is enforced by the OS and by the underlying central processing unit (CPU) hardware so that other users cannot write to or read from the process’ memory. The Crypto-C Toolkit Module resides in one of these “process spaces.” More than one Crypto-C Toolkit Module can reside inside a cryptographic boundary, but the modules operate completely independently and unaware of each other, each in its own “process space”. The operating system performs multi-tasking operations so that only one Crypto-C Toolkit Module is active at any particular moment in time.

Roles and Services

The RSA BSAFE Crypto-C Toolkit Module meets FIPS140-1 Level 1 requirements for Roles and Services. It implements the following two roles: Crypto-Officer role and User role. The Crypto-C Toolkit Module does not support user identification or authentication for these roles.

Only one role may be active at a time and the Crypto-C Module does not allow concurrent operators. API functions can also only be executed one at a time. The use of roles is enforced by the BSAFE Crypto-C Module and not by an external policy.

These roles restrict access to the operation of the BSAFE Crypto-C Toolkit Module. In the BSAFE Cryptographic Toolkit Module, the roles are defined per the FIPS140-1 standard as follows:

- ✓ A **User** is any entity that can operate a client process that uses the Crypto-C Toolkit Module's API. When the User is active, the Crypto-C Toolkit Module is in the **User State**.

- ✓ A **Crypto Officer** is any entity that can operate a client process that uses the Crypto-C API, initiate self-tests, and review the pass-fail results of each self-test. When the Crypto Officer is active, the Crypto-C Toolkit Module is in the **Crypto Officer State**.

- ✓ There is no **Maintenance** role.

The Crypto-Officer is responsible for initiating self-tests and reviewing the status of each test. The User normally operates client processes that use the services of the Crypto-C Module by making API calls to the module from within the same process space. The Crypto Officer can run self-tests with the knowledge that self-test results cannot be accidentally exposed while in the User State. The User is assured that self-tests cannot be initiated during the normal operation of the Module.

It is important to realize that when a transition is made from a User State to a Crypto Officer State all data objects, intermediate states, and key values within the User's or Crypto Officer's process space are actively zeroized. This provides you with a handy way of partitioning access to the Crypto-C Module without having to terminate your application in order to restart your application.

When the Crypto-C Toolkit Module finishes its startup validation and self-tests, it transitions by default into the User Role. A "User" is the owner of a calling process that is operating in the User Role. This is the normal mode in which your application will operate. In this discussion, the User can be thought of as an application program that is making calls to the Crypto-C Toolkit Module.

The User accesses the Crypto-C Toolkit Module API through the use of application programs that are written by application software developers. Application programs are then constructed by developers to import the needed DLL interface definitions and to make API calls to the BSAFE Crypto-C Module. The DLL import library "bsafecsm.lib" is provided along with the Crypto-C Toolkit Module "bsafecsm.dll" that very purpose. All API algorithm objects (AI) and key objects (KI) that are described in the *RSA BSAFE Crypto-C Library Reference Manual* may be used by either the User or the Crypto Officer. The exceptions to the rule are calls that use the "BHAPI" hardware interface. The Crypto-C Module is self-contained and does not rely on underlying cryptographic hardware.

When the Crypto-C Module is started, before the User State is entered, both the Crypto-C Module and the calling User Module are verified for structural integrity and a self-test battery is run to verify the cryptographic functions.

What a User Can Do

- Call the Crypto-C API to perform encryption operations.
- Operate the following API commands:

```
int setOfficerSignInState();
int signInStateIsOfficer();
int signInStateIsUser();
int signInStateIsDisabled();
```

  Prototypes for calling these commands are available in the C-language file signin.h.

- Enter the Crypto Officer State by calling setOfficerSignInState.

What a Crypto Officer Can Do

- Call the Crypto-C API to perform encryption operations.
- Initiate self-test operations by calling the API command runSelfTests.
- View self-test data by calling getSelfTestStatus.
- Operate the following API commands:

```
int setUserSignInState();
int signInStateIsOfficer();
int signInStateIsUser();
int signInStateIsDisabled();
```

  Prototypes for calling these commands are available in the C-language file signin.h

Key Management

The RSA BSAFE Crypto-C Toolkit Module provides cryptographic algorithm support for FIPS140-1 Level 1 requirements for Key Management.

Protocol Support
The Crypto-C Module provides the "low-level" cryptographic functionality that is necessary for implementing key exchange protocols. It does *not* implement the key exchange protocols themselves. The User can export or import keys by writing additional code to do that. When he/she does, the code that imports or exports the keys will reside in the same Windows 'NT "process space" as the Module, assuring safe access between any plain text key data and the Module. To obtain FIPS140-1 validation of the additional application, the user must then distribute keys in a manner that is compatible with the rules of FIPS140-1. The Crypto-C Module only *supports* the manual and electronic distribution of symmetric or asymmetric keys.

That means that it is the User's responsibility to select FIPS140-1 compliant algorithms to perform the key exchange. It is the User's responsibility to understand which algorithms are FIPS-approved and which are not. This information is available in the *RSA BSAFE Crypto-C Library Reference Manual*. The NIST web site also lists approved implementations of NIST-approved cryptographic algorithms.

Key Generation
The Crypto-C Module supports generation of the DSA and RSA public and private keys through user services and employs a FIPS-approved key generation method.

Symmetric keys are not generated by the Crypto-C Module.

The Crypto-C Module supports the entry of electronic keys. However, all keys are stored and used in the Crypto-C Module only for immediate use by a cryptographic process. The Crypto-C Module supports electronic key distribution if a NIST-approved commercial protocol is used. Such protocols employ various secret key, RSA public key and Diffie-Hellman encryption algorithms to properly secure the distribution of electronic keys.

Key Storage
The Crypto-C Module does not provide long-term cryptographic key storage. If the User stores keys, the User is responsible for storing keys in a manner that is compatible with the rules of FIPS140-1.

Two special purpose keys are stored in the Crypto-C Module. There is a single secret cryptographic key that is embedded in the Module as well as a single public DSA key. The secret key is used to decrypt the DSA public key, which is in turn used to validate the Module's integrity. The secret key is zeroized immediately after use and the decrypted public DSA key is zeroized immediately after its use.

If the User stores keys, the User is responsible for storing keys in a manner that is compatible with the rules of FIPS140-1. That also means that it is the User's responsibility to select FIPS140-1 compliant algorithms to perform the key exchange. It is the User's responsibility to understand which algorithms are FIPS-approved and which are not. This information is available in the *RSA BSAFE Crypto-C Library Reference Manual*.

Zeroization of Keys

The Crypto-C Module loads cryptographic keys for use by the User's client process. When the User's process is finished, the keys are zeroized before the client process detaches. The term "client process" refers to the unique memory, CPU time, and other assets that are assigned to a particular task in the host PC.

When a User's process exits or aborts for any reason, the BSAFE Crypto-C Module actively zeroizes all data objects, intermediate states, and key values within the User's process space. In particular, this includes any data that was used with FIPS-based algorithms.

All keys are zeroized from memory prior to unloading the Crypto-C Module and returning its memory to the operating system. The Crypto-C Module zeroizes all keys from computer memory when one of the following occurs:

- Self-tests are initiated
- Transition from User Role to Crypto Officer Role
- Transition from Crypto Officer Role to User Role
- An Error is detected in a cryptographic algorithm
- Just before the Crypto-C Module Exits, or Terminates

Protection of Keys

The Windows ™ 'NT internal memory manager allocates a unique memory space for each algorithm from the User's process space. This process is enforced in the hardware of the Intel Pentium Microprocessor where the OS maintains memory-mapping information. Additionally, the Module prevents data structures that have been allocated to FIPS-approved algorithms from being accessed by non-FIPS algorithms. This is enforced by an internal type check for both key and IV data structures and for intermediate algorithm structures. That check is made every time an operation is performed. "Spoofing" is also prevented because internal checks prevent multiple instances of the same object type.

Any data structures allocated by the Module for use by any algorithm, notably FIPS-approved algorithms, are zeroized before they are released back to the process memory space. This provides a logical partition between the data structures for FIPS-approved and non-FIPS-approved algorithms.

Channel Definitions

FIPS140-1 defines a cryptographic boundary and also channels, through which information is allowed to enter and leave the cryptographic boundary. Defining channels can be straightforward for developers of hardware modules, but developers of software modules are faced with the task of choosing an appropriate set of channel definitions.

FIPS140-1 requires the definition of Data Input/Output and Command/Status channel interfaces. The Crypto-C Module defines these interfaces through the Crypto-C Module's exported DLL library API. The API provides the means to Input and Output Data and to determine the status of the Module. The Data Input/Output interface and the Status interface are active only during the User and Crypto Officer States.

The FIPS140-1 Control interface is used to initiate the Crypto-C Module. It is activated by the operating system when your program asks the Windows ™ 'NT OS to start up the Crypto-C Module. It is otherwise only active during the User and Crypto Officer States when commands are issued via the API in the form of procedure calls. A selected call initiates a specific action. That constitutes "control".

Self-Test

Power-up tests include the known-answer test for the DSA cryptographic algorithm, SHA-1 cryptographic algorithm and DES and TDES cryptographic algorithms and software/firmware digital signature verification. Conditional tests check for failures on an on-going basis. They include the continuous random number generator test as well as pair-wise tests.

An internal failure, such as is detected by a power-up or initiated self-test or by a continuity test will result in a Fatal Error State. A Fatal Error State causes all confidential keying information or other critical data structures to be erased before they are released from the User's process space. An error code that identifies the Fatal Error State is passed to the operating system.

No confidential keying information or other critical data can be received, stored, or transmitted while self-tests are in operation.

Startup Self-Tests

Self-tests are executed on an automatic, mandatory basis at the startup of each User process. That means that for each program that you write, when the program calls the Crypto-C Module for the first time, a delay will occur while self-tests are run. This will happen even if another program is running another "instance" of the Module.

If any of the self-tests fail, the Module will not stay loaded in memory and it will not enter either the User or Crypto Officer State. The module is able to "dump" itself from memory first zeroizing all of the internal data structures, including keying and intermediate data, and then make an "exit" to Windows™ 'NT. Windows™ 'NT then releases the data that was associated with the Module back to system memory.
If any of the self-tests fail, the Module will enter a Fatal Error State. The Fatal Error Status reason code is then passed via the exit command to the operating system.

Conditional Self-Tests
All pseudo-random data that is generated by a FIPS-approved Secure Hash Algorithm (FIPS180-1) is tested to ensure that only non-repeating data is generated (for example, a “stuck” Pseudo-Random Number Generator will be detected.)

All pseudo-random data that is generated by non-FIPS-approved hash algorithms (MD2, MD2x, MD5, and MD5x) is also tested for continuity.

Pair-wise Self-Tests
All FIPS-approved public/private key pairs for the DSA and rDSA (FIPS186-2) algorithms are tested for pair-wise consistency before they are returned to the caller.

All non-FIPS public/private key pairs that are generated for non-FIPS algorithms (e.g. RSA, Elliptic Curve) are tested for pair-wise consistency before they are returned to the caller.

Tests on Imported Parameters
Imported parameters (P, Q, G, X and Y) for the DSS (FIPS186) algorithm are tested for validity. If the imported parameter is invalid, the Module will neither store it nor permit its use in the Module. In addition, a non-zero error will be returned to the User.

Operating Modes

FIPS 140-1 Mode
The Module may run in FIPS140-1 Mode or in non-FIPS140-1 Mode. To run in FIPS140-1 Mode, the Module must use only FIPS-approved cryptographic algorithms as specified by FIPS140-1.

Non-FIPS 140-1 Mode
When the Module is using any non-FIPS-approved cryptographic algorithms, it is in non-FIPS140-1 Mode.

FIPS-Approved Algorithms
It is the User’s responsibility to understand which algorithms are FIPS-approved and which are not. This information is available in *RSA BSAFE Crypto-C Library Reference Manual*. NIST also supports a web site that lists approved implementations of NIST-approved cryptographic algorithms.

BSAFE Crypto-C implements a number of FIPS-approved algorithms. These interfaces are documented fully in the *RSA BSAFE Crypto-C Library Reference Manual*. FIPS compliant Algorithm Object (AI) interfaces are provided by the Crypto-C API are as follows:

Table II, Algorithm Object Interfaces (AI's) for FIPS-Approved Algorithms

Algorithm Object Interfaces (AI's) for FIPS-Approved Algorithms
AI_CBC_IV8
AI_DES_CBC_BSAFE1
AI_DES_CBC_IV8
AI_DES_CBCPadBER
AI_DES_CBCPadIV8
AI_DES_CBCPadPEM
AI_DSA
AI_DSAKeyGen
AI_DSAParamGen
AI_DSAWithSHA1
AI_DSAWithSHA1_BER
AI_FeedbackCipher (DES Modes Only)
AI_SHA1
AI_SHA1_BER
AI_SHA1WithDES_CBCPad
AI_SHA1WithDES_CBCPadBER
AI_SignVerify (rDSA)
AI_RSAStrongKeyGen (rDSA Key Generation)
AI_X931Random (X9.31 compliant, used with AI_RSAStrongKeyGen)

Some of these interfaces can be used together with several other algorithms, not all of which might be FIPS approved. A User Module that restricts its use to FIPS-approved algorithms can access the Crypto-C Module and claim that it is operating in "FIPS 140-1 mode". A User Module that uses any of the above-listed interfaces to access non-FIPS-approved algorithms is by definition no longer in "FIPS 140 mode". It is the user's responsibility to craft his application in such a way as to warn potential operators of this hazard. NIST also supports a web site that lists approved implementations of NIST-approved cryptographic algorithms.

Page: 4
[JH1] I think they meet level 3 requirements right?