RSA Security, Inc.

RSA™ BSAFE®
Crypto-C

RSA BSAFE
Crypto-C
Version 5.2.1

FIPS 140-1 Non-Proprietary Security Policy

Level 1 Validation

Revision 1.0, May 2001

Table of Contents

1 Introduction

1.1 Purpose

This is a Non-Proprietary Federal Information Processing Standard (FIPS) 140-1 Security Policy for the RSA BSAFE™ Crypto-C® Toolkit Module (Crypto-C Module). This Security Policy was produced as part of the FIPS 140-1 level 1 validation of the Crypto-C Module version 5.2.1.

Previous versions of the Crypto-C Module have been validated against FIPS 140-1 requirements, and this Security Policy builds upon and updates those provided for versions 4.11 and 4.3.1. This Security Policy describes how the 5.2.1 version of the Crypto-C Module meets all Level 1 requirements detailed in FIPS 140-1, and how to securely operate the module in compliance with these requirements.

1.2 References

This document describes the Crypto-C Module using FIPS 140-1 terminology for purposes of a FIPS 140-1 validation effort. For non-FIPS related information on the Crypto-C Module, please refer to the RSA web site below, or the RSA BSAFE ® Crypto-C Cryptographic Components for C User's Manual.

FIPS 140-1 (Federal Information Processing Standards Publication 140-1 -- *Security Requirements for Cryptographic Modules*) details U.S. Government requirements for cryptographic modules. More information about the FIPS 140-1 standard and validation program is available on the NIST web site at http://www.nist.gov/fips140-1.

For information on FIPS-approved algorithms please visit the NIST web site at http://csrc.nist.gov/cryptval/.

For more information on RSA Security, the BSAFE line of products, and other RSA Security products, please visit the RSA web site at http://www.rsa.com/.

1.3 Document Organization

The Security Policy document is one part of the complete FIPS 140-1 Submission Package. In addition to this document, the complete Submission Package contains:

- Vendor evidence
- Finite state machine
- Source code
- Other supporting documentation

The first section of this document provides an overview and introduction to the Security Policy. Section 2 describes the module, and how it meets FIPS 140-1 requirements, and Section 3 gives details for operating securely in a FIPS 140-1 compliant manner.

Corsec Security, Inc. produced this Security Policy and other Certification Submission Documentation under contract to RSA Security, Inc. With the exception of this Non-Proprietary

Security Policy, the FIPS 140-1 Certification Submission Documentation is RSA Security-proprietary and is releasable only under appropriate non-disclosure agreements. For access to these documents, please contact RSA Security, Inc.

2 RSA BSAFE Products

With more than a half billion RSA BSAFE-enabled applications being used worldwide to build strong encryption and authentication, Crypto-C is the software developer's tool of choice for security. RSA BSAFE Crypto-C allows state-of-the-art privacy and authentication features to be built into virtually any application, with new, optimized performance. Using Crypto-C means never compromising security for performance or the time it takes the development of your critical applications to reach the market.

RSA BSAFE is a suite of products that provide not only core cryptographic functionality but also out-of-the-box security protocol software components like RSA BSAFE SSL-C/J, RSA BSAFE WTLS-C, RSA BSAFE CERT-C/J and RSA BSAFE S/MIME-C. Each RSA BSAFE developer's kit is a complete software development environment providing comprehensive building-block security components for creating fail-safe software applications.

2.1 The RSA BSAFE Crypto-C Toolkit Module

RSA BSAFE Crypto-C is at the heart of the RSA BSAFE product line. Developers can make use of RSA BSAFE Crypto-C in three convenient formats: a static toolkit (an import library – "bsafe521.lib"), a dynamic toolkit (a dynamic link library (DLL) – csm521.dll), and licensed source code (contact to RSA Security for licensing terms). The second option provides a complete module evaluated against FIPS 140-1 requirements. This module is called the RSA BSAFE Crypto-C Toolkit Module (Crypto-C Module), and incorporates RSA BSAFE Crypto-C into a 32-bit Windows™ 98 compatible DLL that has been validated against all Level 1 FIPS 140-1 requirements.

RSA BSAFE Crypto-C is classified as a multi-chip standalone module for FIPS 140-1 purposes. As such, the Crypto-C Module must be evaluated upon a particular operating system and computer platform. The actual cryptographic boundary thus includes the Crypto-C Module running upon an IBM-compatible PC running the Windows™ 98 Operating System (OS). The Crypto-C Module running on this platform was validated as meeting all FIPS 140-1 level 1 physical security and operating system requirements. The Crypto-C Module is packaged in a single DLL, csm521.dll, which contains all the code for the module.

RSA BSAFE Crypto-C also runs upon many other platforms including Windows '95, '98, NT, and 2000, Sun/Solaris, HP-UX, Linux, and AIX; however, the Crypto-C Module was not implemented and tested upon each of these platforms as part of this effort. RSA BSAFE Crypto-C ports on many other platforms, as well as assembly and custom algorithm optimizations, are available.

2.2 Module Interfaces

As a multi-chip standalone module, the Crypto-C Module's physical interfaces consist of the keyboard, mouse, monitor, serial ports, network adapters, etc. However, the underlying logical interface to the Crypto-C Module is a C-language Application Program Interface (API) documented in the *RSA BSAFE Crypto-C Library Reference Manual*. The module provides for Control Input with the exported DLL library API calls. Data Input and Output are provided in

the variables passed with API calls, and Status Output is provided in the returns and error codes that are documented for each call.

The Crypto-C Module is accessed from C/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 (see section 2.3 for added functions), the same API calls are used in the BSAFE Crypto-C Toolkit Static Library and the Crypto-C Module. In addition, use of the Crypto-C Module enables OEM software developers to construct Crypto-C based applications that automatically rely on a FIPS 140-1 validated module.

2.3 Roles and Services

The RSA BSAFE Crypto-C Toolkit Module meets all FIPS140-1 level 1 requirements for Roles and Services, implementing both a Crypto-Officer role and a User role. As allowed by FIPS 140-1, the Crypto-C 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 Crypto-C Module and not by an external policy. These roles restrict access to the operation of the RSA BSAFE Crypto-C Toolkit Module. The roles are defined per the FIPS140-1 standard as follows:

- The **User** role is assumed by any entity that can operate a client process (an operator) accessing the Crypto-C Module's API. Thus, an operator in the User role normally operates client processes using routine services of the Crypto-C Module, and makes API calls to the module from within the same process space. User role services include:

    - All Crypto-C API functions to perform cryptographic operations
    - **setOfficerSignInState** function call
    - **signInStateIsOfficer** function call
    - **signInStateIsUser** function call
    - **signInStateIsDisabled** function call
    - **setOfficerSignInState** function call to assume Crypto Officer role

- Any operator who has successfully invoked the **setOfficerSignInState** function assumes the **Crypto Officer** role. The Crypto Officer role has responsibility for initiating on-demand self-tests for diagnostic purposes, and reviewing the status of each test. The Crypto Officer role is entered using the **setOfficerSignInState** function. Crypto Officer role services include:

    - All Crypto-C API functions to perform cryptographic operations
    - **runSelfTests** function call
    - **getSelfTestStatus** function call
    - **signInStateIsOfficer** function call
    - **signInStateIsUser** function call

- **signInStateIsDisabled** function call
- **setUserSignInState** function call

Prototypes for non-Crypto-C API functions are available in the C-language file signin.h and selftest.h.

The Crypto Officer can run self-tests with the knowledge that self-test results cannot be accidentally exposed while in the User role. An operator in the User role is assured that self-tests cannot be initiated during the normal operation of the Module. It is important to understand 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 means of partitioning access to the Crypto-C Module without having to terminate your application in order to assume a different role.

When the Crypto-C Toolkit Module finishes its startup validation and self-tests, it transitions by default into the User Role. An operator in the User Role is the owner of a calling process calling the API, which is the normal mode of operation for the module. In this discussion, an "application" is typically an application program making calls to the Crypto-C Toolkit Module.

The operator accesses the Crypto-C Toolkit Module API through the use of applications that are written by application software developers. Developers import the required DLL interface definitions and to make API calls to the BSAFE Crypto-C Module construct applications. The DLL import library bsafe521.lib is provided along with the Crypto-C Toolkit Module csm521.dll for that purpose. Either the User or the Crypto Officer roles may use all API algorithm objects and key objects that are described in the RSA BSAFE Crypto-C Library Reference Manual. 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.

2.4 Cryptographic Key Management

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

2.4.1 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 '98 "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.

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.

2.4.2 Key Generation

The Crypto-C Module supports generation of the DSA public and private keys through user services and employs a FIPS-approved key generation method. The Crypto-C Module does not generate symmetric keys.

2.4.3 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 acceptable under 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.*

2.4.4 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.

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

2.4.5 Protection of Keys

The Windows ™ 98 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.

2.4.6 Cryptographic Algorithms

The Crypto-C Module supports a wide variety of cryptographic algorithms. FIPS 140-1 requires that FIPS-defined algorithms be used whenever there is an applicable FIPS standard of that type. Thus, as the following table summarizes, only a subset of the algorithms provided by the Crypto-C Module may be used in a FIPS 140-1 Mode. For more information on the FIPS 140-1 Mode, please refer to Section 3.

Table 1 – Algorithms supported by the Crypto-C Module

Type	Algorithm	FIPS
Symmetric Cipher		
	AES (Advanced Encryption Standard)	
	DES (Data Encryption Standard)	FIPS 46-3
	Triple DES	FIPS 46-3
	DESX	
	RC2® block cipher	
	RC4® stream cipher	
	RC5™ block cipher	
	RC6™ block cipher	
Message Digest		
	MD (Message Digest)	
	MD2	
	MD5	
	SHA1 (Secure Hashing Algorithm)	FIPS 180-1
Message Authentication		
	HMAC	
Random-Number Generation		
	MD2	
	MD5	
	SHA1	FIPS 186-2
	ANSI X9.31	FIPS 186-2
Public-Key Algorithm		
	RSA Public Key Cryptosystem	
	Diffie-Hellman Key Agreement	
Digital Signature		
	DSA	FIPS 186-2
	RSA Digital Signatures	FIPS 186-2
Elliptic Curve Public-Key Algorithm		
	Elliptic Curve Digital Signature Algorithm (ECDSA)	FIPS 186-2
	Elliptic Curve Diffie-Hellman Key Agreement	
	Elliptic Curve Authenticated Encryption Scheme	

	(ECAES)	
Secret Sharing		
	Bloom-Shamir Secret Sharing	

All FIPS-defined algorithms in the Crypto-C Module have been separately validated through a FIPS-approved testing program. (A FIPS-approved testing program does not yet exist for some newer algorithms such as ECDSA.)

2.5 Self-Tests

When the Crypto-C Module is started, it executes self-tests to ensure correct operation of the cryptographic algorithms, and integrity of the module. Should any self-test fail, the module transitions to 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.

Self-tests include a known-answer test for the DSA cryptographic algorithm, SHA-1 cryptographic algorithm, DES and Triple DES cryptographic algorithms, and a software/firmware digital signature verification. Additionally, conditional tests check for failures on an on-going basis. These tests include a continuous random number generator test as well as pair-wise consistency tests upon asymmetric key generation.

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

2.5.1 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™ 98. Windows™ 98 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.

2.5.2 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, MD5x) is also tested for continuity.

2.5.3 Pair-wise Self-Tests

All FIPS-approved public/private key pairs for the DSS (FIPS186) algorithm 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.

3 Secure Operation of the Module

RSA BSAFE Crypto-C Toolkit Module does not require any special configuration to operate in a FIPS 140-1 mode. RSA BSAFE offers a wide array of the most advanced public domain and proprietary cryptographic algorithms available. However, FIPS 140-1 requires that only FIPS-approved algorithms be used when operating a module in a FIPS 140-1 mode. Therefore, to operate the Crypto-C module in a FIPS 140-1 Mode, only the FIPS-approved algorithms listed in section 2.4.6 may be used.

It is the User's responsibility to understand which algorithms are FIPS-approved and which are not. This information is further explained in *RSA BSAFE Crypto-C Library Reference Manual*. NIST also supports a web site (referenced in section 1.2) that lists approved implementations of NIST-approved cryptographic algorithms. In addition, Table 2 lists the FIPS-compliant Algorithm Object (AI) interfaces that are used when making API calls to the Crypto-C Module.

Table 2 – Algorithm Object Interfaces (AIs) for FIPS-Approved Algorithms

AI_CBC_IV8 (when used with AI_DES_*XXX* AIs)
AI_DES_CBC_BSAFE1
AI_DES_CBC_IV8
AI_DES_CBCPadBER
AI_DES_CBCPadIV8
AI_DES_CBCPadPEM
AI_DES_EDE3_CBC_IV8
AI_DES_EDE3_CBCPadBER
AI_DES_EDE3_CBCPadIV8
AI_DSA
AI_DSAKeyGen
AI_DSAParamGen
AI_DSAWithSHA1
AI_DSAWithSHA1_BER
AI_ECAcceleratorTable
AI_ECBuildAcceleratorTable
AI_ECBuildPubKeyAccelTable
AI_EC_DSA
AI_EC_DSAWithDigest
AI_ECKeyGen
AI_ECParameters
AI_ECParametersBER
AI_ECParamGen
AI_ECPubKey
AI_ECPubKeyBER
AI_FeedbackCipher (DES Modes Only)
AI_PKCS_RSAPrivate

AI_PKCS_RSAPrivateBER
AI_PKCS_RSAPrivatePEM
AI_PKCS_RSAPublic
AI_PKCS_RSAPublicBER
AI_PKCS_RSAPublicPEM
AI_RESET_IV
AI_RSAKeyGen
AI_RSAMultiPrimeKeyGen
AI_RSAPrivate
AI_RSAPrivateBSAFE1
AI_RSAPublic
AI_RSAPublicBSAFE1
AI_RSAStrongKeyGen
AI_SHA1
AI_SHA1_BER
AI_SHA1WithDES_CBCPad
AI_SHA1WithDES_CBCPadBER
AI_SHA1Random
AI_SHA1WithRSAEncryption
AI_SHA1WithRSAEncryptionBER
AI_SignVerify
AI_X931Random

Some of these AI interfaces support the use of multiple algorithms, both FIPS-approved and non-FIPS approved. An operator (or calling application) 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". An operator that uses AIs to access non-FIPS-approved algorithms is by definition no longer in "FIPS 140 mode". It is the user's responsibility to craft an application in such a way as to warn potential operators of this hazard. The pipelined and interleaved Triple DES modes available via AI_FeedbackCipher may not be used when operating the module in a FIPS-compliant manner.

To access FIPS 140-1 compliant random number generation and key generation capabilities it is necessary to use the SAFETY_ALGORITHM_CHOOSER that is exported by the csm521.dll. This algorithm chooser has been "hooked" such that FIPS compliant functionality is exercised in place of default functionality for the following AMs associated with FIPS-approved algorithms: AM_SHA_RANDOM, AM_DSA_KEY_GEN, AM_RSA_KEY_GEN, AM_RSA_STRONG_KEY_GEN, AM_ECFP_KEY_GEN and AM_ECF2POLY_KEY_GEN.