26.11.14

RSA BSAFE® Crypto-C Micro Edition

Version 3.0.0.1, 3.0.0.14, 3.0.0.15, 3.0.0.19, 3.0.0.21, and 3.0.0.23 Security Policy

This is a non-proprietary security policy for RSA BSAFE Crypto-C Micro Edition (Crypto-C ME) 3.0.0.1, 3.0.0.14, 3.0.0.15, 3.0.0.19, 3.0.0.21, and 3.0.0.23 security software.

Contents:

Preface

This security policy describes how Crypto-C ME meets the security requirements of FIPS 140-2. The security policy also describes how to securely operate Crypto-C ME in a FIPS 140-2 compliant manner.

The Crypto-C ME software development toolkit enables developers to incorporate cryptographic technologies into applications. Crypto-C ME security software is designed to help protect sensitive data as it is stored, using strong encryption techniques that ease integration with existing data models. Using the capabilities of Crypto-C ME software in applications helps provide a persistent level of protection for data, lessening the risk of internal, as well as external, compromise.

FIPS 140-2 (Federal Information Processing Standards Publication 140-2 - Security Requirements for Cryptographic Modules) details the United States Government requirements for cryptographic modules. More information about the FIPS 140-2 standard and validation program is available on the NIST website (http://csrc.nist.gov/groups/STM/cmvp/index.html).

References

This document deals only with operations and capabilities of the Crypto-C ME Module and Crypto-C ME's use of this module in terms of a FIPS 140-2 Cryptographic Module security policy.

For more information about Crypto-C ME, the Crypto-C ME Module, and the entire RSA BSAFE product line, see:

- Information on the full line of RSA products and services is available at www.emc.com/domains/rsa/.
- RSA BSAFE product overviews are available at www.emc.com/security/rsa-bsafe.htm.
- Answers to technical or sales related questions are available at www.emc.com/support/rsa/.

Terminology

In this document, the term Cryptographic Module denotes the Crypto-C ME FIPS 140-2 validated Cryptographic Module.

The Crypto-C ME Module is also referred to as:

- The Cryptographic Module
- The module.

Document Organization

This Security Policy explains the Cryptographic Module's FIPS 140-2 relevant features and functionality. This document comprises the following sections:

- This section, "Preface" on page 2 provides an overview and introduction to the Security Policy.
- "Crypto-C ME Cryptographic Toolkit" on page 4 describes Crypto-C ME and how it meets FIPS 140-2 requirements.
- "Secure Operation of Crypto-C ME" on page 14 specifically addresses the required configuration for the FIPS 140-2 mode of operation.
- "Services" on page 19 lists the functions of Crypto-C ME.
- "Acronyms" on page 22 lists the acronyms and definitions used in this document.

1 Crypto-C ME Cryptographic Toolkit

The Crypto-C ME software development toolkit enables developers to incorporate cryptographic technologies into applications. Crypto-C ME security software is designed to help protect sensitive data as it is stored, using strong encryption techniques that ease integration with existing data models. Using the capabilities of Crypto-C ME software in applications helps provide a persistent level of protection for data, lessening the risk of internal, as well as external, compromise.

The features of Crypto-C ME include the ability to optimize code for different processors, and specific speed or size requirements. Assembly-level optimizations on key processors mean that Crypto-C ME algorithms can be used at increased speeds on many platforms.

Crypto-C ME offers a full set of cryptographic algorithms including asymmetric key algorithms, symmetric key block and stream algorithms, message digests, message authentication, and Pseudo Random Number Generator (PRNG) support. Developers can implement the full suite of algorithms through a single Application Programming Interface (API) or select a specific set of algorithms to reduce code size or meet performance requirements.

Note: When operating in a FIPS 140-2-approved manner, the set of algorithm implementations is not customizable.

1.1 Cryptographic Module

Crypto-C ME is classified as a multi-chip standalone cryptographic module for the purposes of FIPS 140-2. As such, Crypto-C ME must be tested on a specific operating system and computer platform. The cryptographic boundary includes Crypto-C ME running on selected platforms running selected operating systems while configured in “single user” mode. Crypto-C ME was validated as meeting all FIPS 140-2 Level 1 security requirements, including cryptographic key management and operating system requirements.

Crypto-C ME is packaged as a set of dynamically loaded modules or shared library files that contain the module's entire executable code. The Crypto-C ME toolkit relies on the physical security provided by the host PC in which it runs.

For FIPS 140-2 validation, Crypto-C ME is tested on the following platforms:

- NetBSD® (tested with Crypto-C ME 3.0.0.15):
  - NetBSD v2.0.3 x86 (32-bit)
  - NetBSD v2.0.3 MIPS RM7035c (32-bit).
- Timesys™ (tested with Crypto-C ME 3.0.0.14 and 3.0.0.19):
  - Linux 2.6.26.8-rt16 PowerPC® (32-bit).
- Timesys (tested with Crypto-C ME 3.0.0.21):
  - Linux 2.6.33.9-rt ARM® v7 (MV78230) (32-bit)
  - Linux 3.0.0-rt ARM v7 (PJ4B-MOP) (32-bit)

- Linaro (tested with Crypto-C ME 3.0.0.23):
  - Linux (kernel 3.10.33) ARM Cortex A7 Dual Core (MB86S73) (32-bit)
- Red Hat® (tested with Crypto-C ME 3.0.0.1):
  - Enterprise Linux® AS 4.0 x86 (32-bit) with LSB 3.0.3.
- Windows® (tested with Crypto-C ME 3.0.0.1):
  - Vista™ Ultimate x86 (32-bit) – Visual Studio 2005 SP1 (MD option)
  - XP Professional SP2 x86 (32-bit) – Visual Studio 2005 SP1 (MT option).

Note: Compliance is maintained on all of the above platforms for which the binary executable remains unchanged.

For resolution of the "Multi User" modes issue, see the NIST document, *Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program*, located at http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf.

The following table lists the certification levels sought for Crypto-C ME for each section of the FIPS 140-2 specification.

Table 1 Certification Levels

Section of the FIPS 140-2 Specification	Level
Cryptographic Module Specification	3
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Managment	1
EMI/EMC	1
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	1
Overall	1

1.1.1 Configuring Single User Mode

This section describes how to configure single user mode for the different operating system platforms supported by Crypto-C ME.

NetBSD, Timesys Linux, Linaro Linux, and Red Hat Linux

To configure single user mode for systems running a NetBSD operating system:

1. Log in as the `root` user.
2. Edit `/etc/passwd` and `/etc/shadow` to remove all the users except `root` and the pseudo-users (daemon users). Make sure the password fields in `/etc/shadow` for the pseudo-users are either a star (`*`) or double exclamation mark (`!!`). This prevents login as the pseudo-users.
3. Edit `/etc/nsswitch.conf` so that files is the only option for `passwd`, `group`, and `shadow`. This disables the Network Information Service (NIS) and other name services for users and groups.
4. In the `/etc/xinetd.d` directory, edit `rexec`, `rlogin`, `rsh`, `rsync`, `telnet`, and `wu-ftpd` as required, setting the value of `disable` to `yes`.
5. Reboot the system for the changes to take effect.

Microsoft Windows

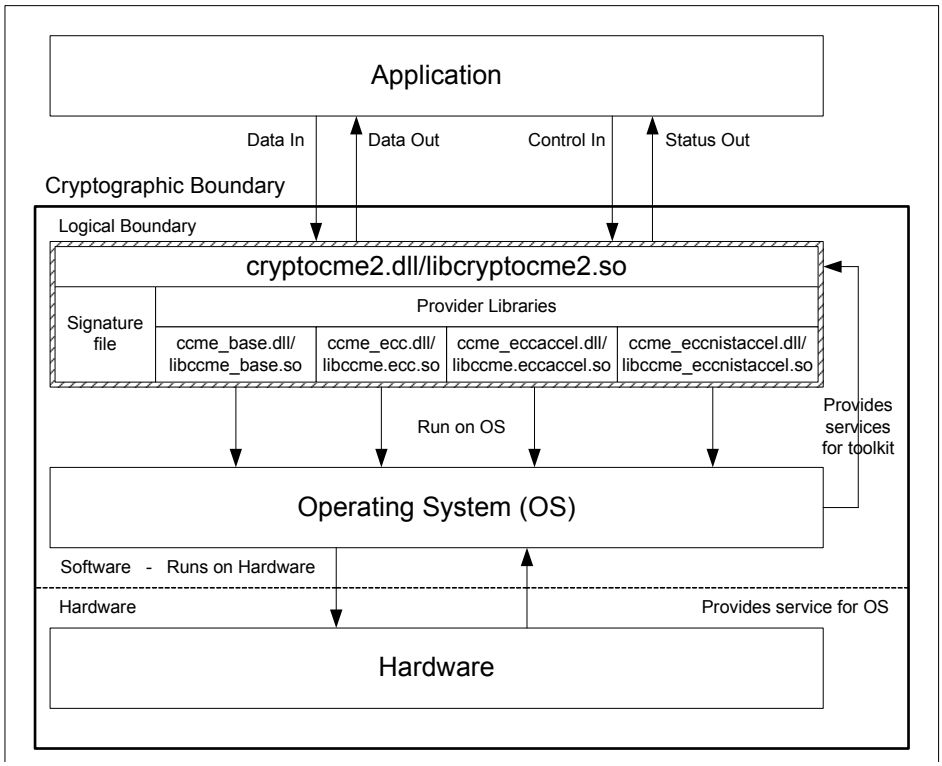
To configure single user mode for systems running a Microsoft Windows XP Professional SP2 or Windows Vista Ultimate operating system, guest accounts, server services, terminal services, remote registry services, remote desktop services, and remote assistance must be disabled. For detailed instructions on how to perform these tasks, see the Microsoft support site.

1.2 Crypto-C ME Interfaces

Crypto-C ME is validated as a multi-chip, standalone module. The physical cryptographic boundary of the module is the case of the general-purpose computer or mobile device, which encloses the hardware running the module. The physical interfaces for Crypto-C ME consist of the keyboard, mouse, monitor, CD-ROM drive, floppy drive, serial ports, USB ports, COM ports, and network adapter(s).

The logical boundary of the cryptographic module is the set of library files (`libcryptocme2.so`, `libccme_base.so`, `libccme_ecc.so`, `libccme_eccaccel.so`, and `libccme_eccnistaccel.so`) and the signature file that comprise the module. The underlying logical interface to Crypto-C ME is the API, documented in the *RSA BSAFE Crypto-C Micro Edition API Reference Guide*. Crypto-C ME provides for Control Input through the API calls. Data Input and Output are provided in the variables passed with the API calls, and Status Output is provided through the returns and error codes that are documented for each call. This is illustrated in the following diagram.

Figure 1 Crypto-C ME Logical Interfaces



1.3 Roles and Services

Crypto-C ME meets all FIPS 140-2 Level 1 requirements for roles and services, implementing both a User (User) role and Officer (CO) role. As allowed by FIPS 140-2, Crypto-C ME does not support user identification or authentication for these roles. Only one role can be active at a time and Crypto-C ME does not allow concurrent operators.

The following table describes the services accessible by the two roles.

Table 2 Crypto-C ME Roles and Services

Role	Services
Officer	The Officer has access to a super-set of the services that are available to the User. The Officer role can also invoke the full set of self-tests inside the module.
User	The User can perform general security functions, as described in the *RSA BSAFE Crypto-C Micro Edition API Reference Guide*. The User can also call specific FIPS 140-2 module functions as defined in the *API Reference Guide*.

1.3.1 Officer Role

An operator assuming the Officer role can call any Crypto-C ME function. The complete list of the functionality available to the Officer is outlined in "Services" on page 19.

1.3.2 User Role

An operator assuming the User role can use the entire Crypto-C ME API except for `R_FIPS140_self_test_full()`, which is reserved for the Officer. The complete list of Crypto-C ME functions is outlined in "Services" on page 19.

1.4 Cryptographic Key Management

Cryptographic key management is concerned with generating and storing keys, managing access to keys, protecting keys during use, and zeroizing keys when they are not longer required.

1.4.1 Key Generation

Crypto-C ME supports generation of DSA, RSA, Diffie-Hellman (DH) and Elliptic Curve Cryptography (ECC) public and private keys. Also, Crypto-C ME uses a FIPS 186-2 compliant random number generator as well as a Dual Elliptic Curve Deterministic Random Bit Generator (Dual ECDRBG) and HMAC-DRBG in the generation of asymmetric and symmetric keys used in algorithms such as AES, Triple-DES, RSA, DSA, DH, ECC, and HMAC.

1.4.2 Key Storage

Crypto-C ME does not provide long-term cryptographic key storage. If a user chooses to store keys, the user is responsible for storing keys exported from the module.

The following table lists all keys and CSPs in the module and where they are stored.

Table 3 Key Storage

Key or CSP	Storage
Hardcoded DSA public key	Persistent storage embedded in the module binary (encrypted).
Hardcoded AES key	Persistent storage embedded in the module binary (plaintext).
AES keys	Volatile memory only (plaintext).
Triple-DES keys	Volatile memory only (plaintext).
HMAC with SHA-1 and SHA-2 keys (SHA-224, SHA-256, SHA-384, SHA-512)	Volatile memory only (plaintext).
DH public/private keys	Volatile memory only (plaintext).
ECC public/private keys	Volatile memory only (plaintext).
RSA public/private keys	Volatile memory only (plaintext).
DSA public/private keys	Volatile memory only (plaintext).
FIPS 186-2 seed	Volatile memory only (plaintext).
FIPS 186-2 key	Volatile memory only (plaintext).
EC DRBG entropy	Volatile memory only (plaintext).
EC DRBG S value	Volatile memory only (plaintext).
EC DRBG init_seed	Volatile memory only (plaintext).
HMAC DRBG entropy	Volatile memory only (plaintext).
HMAC DRBG V value	Volatile memory only (plaintext).
HMAC DRBG key	Volatile memory only (plaintext).
HMAC DRBG init_seed	Volatile memory only (plaintext).

1.4.3 Key Access

An authorized operator of the module has access to all key data created during Crypto-C ME operation.

Note: The User and Officer roles have equal and complete access to all keys.

1.4.4 Key Protection/Zeroization

All key data resides in internally allocated data structures and can be output only using the Crypto-C ME API. The operating system protects memory and process space from unauthorized access. The operator should follow the steps outlined in the *RSA BSAFE Crypto-C Micro Edition API Reference Guide* to ensure sensitive data is protected by zeroizing the data from memory when it is no longer needed. All volatile keys and CSPs listed in Table 3 are zeroized by unloading the module from memory.

1.5 Cryptographic Algorithms

Crypto-C ME supports a wide variety of cryptographic algorithms. To achieve compliance with the FIPS 140-2 standard, only FIPS 140-2-approved or allowed algorithms can be used in an approved mode of operation.

The following table lists the FIPS 140-2-approved algorithms supported by Crypto-C ME with validation certificate numbers for Crypto-C ME 3.0.0.1, 3.0.0.14, 3.0.0.15, 3.0.0.19, and 3.0.0.21.

Table 4 Crypto-C ME FIPS 140-2-approved Algorithms

	Validation Certificates				
Algorithm	**3.0.0.1**	**3.0.0.14 and 3.0.0.19**	**3.0.0.15**	**3.0.0.21**	**3.0.0.23**
AES ECB, CBC, CFB (128-bit), OFB (128-bit), CTR (128, 192, and 256-bit), and CCM	860	1771	1951	2808	3120
AES GCM (128, 192, 256-bit) and GMAC (128, 192, and 256-bit)	Vendor affirmed	Vendor affirmed	Vendor affirmed	Vendor affirmed	Vendor affirmed
Triple-DES ECB, CBC, CFB (64bit), and OFB (64-bit).	707	1147	1268	1686	1791
Diffie-Hellman, EC-Diffie-Hellman, and EC-Diffie-Hellman with Components	Non-approved[1]	Non-approved[1]	Non-approved[1]	Non-approved[1]	Non-approved[1]
DSA	311	554	623	850	904
ECDSA	98 and 100	239 and 240	281 and 283	491 and 492 503 and 504	565 and 569

Table 4 Crypto-C ME FIPS 140-2-approved Algorithms (continued)

Algorithm	Validation Certificates				
	3.0.0.1	3.0.0.14 and 3.0.0.19	3.0.0.15	3.0.0.21	3.0.0.23
FIPS 186-2 Pseudo Random Number Generator (PRNG) - Change Notice 1, with and without the mod q step	492	943	1027	1274	1330
Dual ECDRBG and HMAC-DRBG	4	122	172	480	632
RSA X9.31, PKCS #1 V.1.5, and PKCS #1 V.2.1, (SHA256 - PSS)	412	887	1012	1470 1489	1591
RSA encrypt and decrypt	Non-approved[1]	Non-approved[1]	Non-approved[1]	Non-approved[1]	Non-approved[1]
SHA-1	855	1555	1713	2356	2578
SHA-224, 256, 384, and 512	855	1555	1713	2356	2578
HMAC-SHA1, SHA224, SHA256, SHA384, and SHA512	477	1040	1177	1759	1959

[1]Allowed in FIPS 140-2 mode for key transport

The following Crypto-C ME algorithms are not FIPS 140-2-approved:

- DES
- MD2
- MD5
- HMAC MD5
- DES40
- RC2
- RC4
- RC5
- ECAES
- ECIES
- PBKDF1 SHA-1
- PBKDF2 HMAC SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512
- RSA PKCS #1 V.2.0 (OAEP)
- Entropy RNG
- OTP RNG.

For more information about using Crypto-C ME in a FIPS 140-2 compliant manner, see “Secure Operation of Crypto-C ME” on page 14.

1.6 Self Tests

Crypto-C ME performs a number of power-up and conditional self-tests to ensure proper operation.

If the power-up self-test fails, the toolkit is disabled and the operation fails. If the ECC provider self-test fails, the provider libraries (`libccme_base.so`, `libccme_ecc.so`, and `libccme_eccaccel.so`) are disabled and the operation fails. The toolkit can only leave the disabled state by reloading the FIPS 140-2 module. If the conditional self-test fails, the operation fails but the toolkit is not disabled.

For self-test failures (power-up or conditional) the library notifies the user through the returns and error codes for the API.

1.6.1 Power-up Self-test

Crypto-C ME implements the following power-up self-tests:

- AES, AES CCM, AES GCM, and AES GMAC Known Answer Tests (KATs)
- DES and Triple-DES KATs
- SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 KATs
- HMAC SHA-1, HMAC SHA-224, HMAC SHA-256, HMAC SHA-384, and HMAC SHA-512 KATs
- RSA sign/verify test
- DSA sign/verify test
- DH, EC-DH, and EC-DH with Components conditional test
- ECDSA sign/verify test
- PRNG (FIPS 186-2, Dual ECDRBG, and HMAC-DRBG) KATs
- Software integrity test.

Power-up self-tests are executed automatically when Crypto-C ME is loaded into memory.

1.6.2 Conditional Self-tests

Crypto-C ME performs two conditional self-tests:

- A pair-wise consistency test each time Crypto-C ME generates a DSA, RSA, or EC public/private key pair.
- A Continuous Random Number Generation (CRNG) test each time the toolkit produces random data, as per the FIPS 186-2 standard. The CRNG test is performed on all approved and non-approved RNGs.

1.6.3 Critical Functions Tests

Depending on operating mode, Crypto-C ME performs the following known answer tests:

- In `R_FIPS140_MODE_FIPS140_SSL` mode, Crypto-C ME performs a known answer test for MD5 and HMAC-MD5.
- In `R_FIPS140_MODE_FIPS140_ECC` mode, Crypto-C ME performs a known answer test for ECAES and ECIES.
- In `R_FIPS140_MODE_SSL_ECC` mode, a known answer test is performed for MD5, HMAC-MD5, ECAES, and ECIES.

1.6.4 Mitigation of Other Attacks

RSA key operations implement blinding, a reversible way of modifying the input data, so as to make the RSA operation immune to timing attacks. Blinding has no effect on the algorithm other than to mitigate attacks on the algorithm. Blinding is implemented through blinding modes, and the following options are available:

- Blinding mode off.
- Blinding mode with no update, where the blinding value is constant for each operation.
- Blinding mode with full update, where a new blinding value is used for each operation.

2 Secure Operation of Crypto-C ME

This section provides an overview of how to securely operate Crypto-C ME in compliance with the FIPS 140-2 standards.

2.1 Crypto Officer and User Guidance

The Crypto Officer and User must only use algorithms approved for use in a FIPS 140 mode of operation, as listed in Table 4 on page 10. The requirements for using the approved algorithms in a FIPS 140 mode of operation are as follows:

- The bit length for a DSA key pair must be 1024 bits.
- Bit lengths for an RSA key pair must be between 1024 and 4096 bits in multiples of 512.
- Bit lengths for an HMAC key must be between 80 and 4096 bits.
- EC key pairs must have named curve domain parameters from the set of NIST-recommended named curves (P192, P224, P256, P384, P521, B163, B233, B283, B409, B571, K163, K233, K283, K409, K571). The module limits possible curves for Dual EC DRBG to P256, P384, and P521 in accordance with SP 800-90.
- When using RSA for key wrapping, the strength of the methodology is between 80 and 150 bits of security.
- The Diffie-Hellman shared secret provides between 80 and 150 bits of encryption strength.
- EC Diffie-Hellman primitives must use curve domain parameters from the set of NIST-recommended named curves. Using NIST-recommended curves, the computed Diffie-Hellman shared secret provides between 80 and 256 bits of encryption strength.
- When using an approved RNG to generate keys, the requested security strength for the RNG must be at least as great as the security strength of the key being generated.

2.2 Roles

If a user of Crypto-C ME needs to operate the toolkit in different roles, then the user must ensure that all instantiated cryptographic objects are destroyed before changing from the Crypto User role to the Crypto Officer role, or unexpected results could occur.

The following table lists the roles a user can operate in.

Table 5 Crypto-C ME Roles

Role	Description
`R_FIPS140_ROLE_OFFICER`	An operator assuming the Crypto Officer role can call any Crypto-C ME function. The complete list of the functionality available to the Crypto Officer is outlined in "Services" on page 19.
`R_FIPS140_ROLE_USER`	An operator assuming the Crypto User role can use the entire Crypto-C ME API except for `R_FIPS140_self_test_full()`, which is reserved for the Crypto Officer. The complete list of Crypto-C ME functions is outlined in "Services" on page 19.

2.3 Modes of Operation

The following table lists and describes the available modes of operation.

Table 6 Crypto-C ME Modes of Operation

Mode	Description
`R_FIPS140_MODE_FIPS140` FIPS 140-2-approved.	Provides the cryptographic algorithms listed in Table 4 on page 10. The default random number generator is the FIPS 186-2 PRNG. This is theCrypto-C ME default mode on start up.
`R_FIPS140_MODE_FIPS140_SSL` FIPS 140-2-approved if used with TLS protocol implementations.	Provides the same algorithms as `R_FIPS140_MODE_FIPS140`, plus the MD5 message digest. This mode can be used in the context of the key establishment phase in the TLSv1 and TLSv1.1 protocol. For more information, see section 7.1 Acceptable Key Establishment Protocols in *Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program* (`http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf`). The implementation guidance disallows the use of the SSLv2 and SSLv3 versions. Cipher suites that include non-FIPS 140-2-approved algorithms are unavailable. This mode allows implementations of the TLS protocol to operate Crypto-C ME in a FIPS 140-2 compliant manner with the FIPS 186-2 PRNG as the default.

Table 6 Crypto-C ME Modes of Operation (continued)

Mode	Description
`R_FIPS140_MODE_FIPS140_ECC` Not FIPS 140-2-approved.	Provides the same algorithms as `R_FIPS140_MODE_FIPS140`, plus ECAES and ECIAS. The random number generator in this mode is the Dual ECDRBG.
`R_FIPS140_MODE_FIPS140_SSL_ECC` Not FIPS 140-2-approved.	Provides the same algorithms as `R_FIPS140_MODE_FIPS140_SSL`, plus ECAES and ECIAS. The random number generator in this mode is the Dual ECDRBG. The same restrictions with respect to protocol versions and cipher suites as in `R_FIPS140_MODE_FIPS140_SSL` apply.
`R_FIPS140_MODE_NON_FIPS140` Not FIPS 140-2-approved.	Allows users to operate Crypto-C ME without any cryptographic algorithm restrictions.
`R_FIPS140_MODE_DISABLED` Not FIPS 140-2-approved.	Indicates that the FIPS140 library is disabled, usually due to an internal or caller's usage error. No future transition into other modes is permitted.

In each mode of operation, the complete set of services, which are listed in this Security Policy, are available to both the Crypto Officer and User roles (with the exception of `R_FIPS140_self_test_full()`, which is always reserved for the Crypto Officer).

Note: Cryptographic keys must not be shared between modes. For example, a key generated in `R_FIPS140_MODE_FIPS140` mode must not be shared with an application running in `R_FIPS140_MODE_NON_FIPS140` mode.

2.4 Operating Crypto-C ME

Crypto-C ME operates in `R_FIPS140_MODE_FIPS140` by default if the Crypto-C ME library is initialized with the `PRODUCT_DEFAULT_RESOURCE_LIST()`. Call `R_FIPS140_get_mode()` to determine the current mode of Crypto-C ME.

When changing the mode of operation to a FIPS-approved mode, the module must be re-initialized with the appropriate product resource list, (`PRODUCT_DEFAULT_RESOURCE_LIST()` or `PRODUCT_FIPS140_SWITCH_RESOURCE_LIST()` for `R_FIPS140_MODE_FIPS140`, or `PRODUCT_FIPS140_SSL_SWITCH_RESOURCE_LIST()` for `R_FIPS140_MODE_FIPS140_SSL`). This ensures that the module is reloaded and all power-up self-tests are properly executed. To change the module to a non-FIPS-approved mode, call `R_FIPS140_set_mode()` with one of the information identifiers listed in Table 6 on page 15.

Note: `R_FIPS140_set_mode()` can only be used when changing to a non-FIPS-approved mode.

After setting Crypto-C ME into a FIPS 140-2-approved mode, Crypto-C ME enforces that only the algorithms listed in Table 4 on page 10 are available to operators. To disable FIPS 140-2 mode, call `R_FIPS140_set_mode()` with the `R_FIPS140_MODE_NON_FIPS140` information identifier.

`R_FIPS140_self_tests_full()` is restricted to operation by the Crypto Officer.

The user of Crypto-C ME links with the static library for their platform, which loads the Crypto-C ME shared or dynamic link master and provider libraries at runtime. For more information, see "FIPS 140-2 Library and Modes of Operation" in the *RSA BSAFE Crypto-C Micro Edition API Reference Guide*.

The current Crypto-C ME role can be determined calling `R_FIPS140_get_role()`. The role can be changed by calling `R_FIPS140_set_role()` with one of the information identifiers listed in Table 5 on page 15.

2.5 Startup Self-tests

Crypto-C ME provides the ability to configure when power-up self-tests are executed. To operate Crypto-C ME in a FIPS 140-2 compliant manner, the default shipped configuration, which executes the self-tests when the module is first loaded, must be used.

For more information about this configuration setting, see the *RSA BSAFE Crypto-C Micro Edition Installation Guide*.

2.6 Random Number Generator

In FIPS 140-2 modes, Crypto-C ME provides a default RNG. For `R_FIPS140_MODE_FIPS140` and `R_FIPS140_MODE_FIPS140_SSL`, Crypto-C ME provides a FIPS 186-2 PRNG for all operations that require the generation of random numbers. For `R_FIPS140_MODE_FIPS140_ECC` and `R_FIPS140_MODE_FIPS140_SSL_ECC`, Crypto-C ME implements a Dual ECDRBG internally.

In all modes, users can choose to use an approved RNG other than the default RNG, including the FIPS 186-2 PRNG (with or without mod q), Dual ECDRBG, or HMAC DRBG when creating a RNG object and setting this object against the operation requiring random number generation (for example, key generation). However, when DSA is used, the RNG used internally is always the FIPS 186-2 Change Notice 1 Option 1 with mod q PRNG.

This module also includes a non-approved Entropy RNG that is used to generate seed material for the approved PRNGs.

2.6.1 PRNG Seeding

In the FIPS 140-2 validated library, Crypto-C ME implements deterministic random number generators that can be called to generate random data. The quality of the random data output from these RNGs depends on the quality of the supplied seeding (entropy). Crypto-C ME provides internal entropy collection (for example, from high precision timers) where possible, but it is strongly recommended to collect entropy from external sources.

This is particularly critical if developing on embedded platforms where there are only limited internal entropy sources available. For more information on seeding PRNGs, see "Randomness Recommendations for Security" in RFC 1750.

The `R_CR_INFO_ID_RAND_ENTROPY_FUNC` identifier specifies that additional entropy be available. `R_CR_INFO_ID_RAND_ENTROPY_FUNC` is set against the `R_CR` object, which encapsulates the random number generator, and takes a callback function that the random number generator then uses to gather additional entropy if needed. For more information about `R_CR_INFO_ID_RAND_ENTROPY_FUNC`, see the *RSA BSAFE Crypto-C Micro Edition API Reference Guide*.

3 Services

The following is the list of services provided by Crypto-C ME. For more information about these functions, see the *RSA BSAFE Crypto-C Micro Edition API Reference Guide*.

```
BIO_append_filename()
BIO_clear_flags()
BIO_clear_retry_flags()
BIO_copy_next_retry()
BIO_debug_cb()
BIO_dump()
BIO_dump_format()
BIO_dup_chain()
BIO_f_buffer()
BIO_f_null()
BIO_find_type()
BIO_flags_to_string()
BIO_flush()
BIO_free()
BIO_free_all()
BIO_get_cb()
BIO_get_cb_arg()
BIO_get_close()
BIO_get_flags()
BIO_get_fp()
BIO_get_retry_BIO()
BIO_get_retry_flags()
BIO_get_retry_reason()
BIO_get_ssl()
BIO_get_state_cb()
BIO_get_state_cb_arg()
BIO_gets()
BIO_method_name()
BIO_method_type()
BIO_new()
BIO_new_file()
BIO_new_fp()
BIO_new_mem()
BIO_open_file()
BIO_pop()
BIO_print_hex()
BIO_printf()
BIO_push()
BIO_puts()
BIO_read()
BIO_read_filename()
BIO_reference_inc()
BIO_reset()
BIO_retry_type()
BIO_rw_filename()
BIO_s_file
BIO_s_mem()
BIO_s_null()
BIO_seek()
BIO_set_bio_cb()
BIO_set_cb()
BIO_set_cb_arg()
BIO_set_close()
BIO_set_flags()
BIO_set_fp()
BIO_should_io_special()
BIO_should_read()
BIO_should_retry()
BIO_should_write()
BIO_state_to_string()
BIO_tell()
BIO_write()
BIO_write_filename()
PRODUCT_DEFAULT_RESOURCE_LIST()
PRODUCT_FIPS140_ECC_SWITCH_RESOURCE_LIST()
PRODUCT_FIPS140_SSL_ECC_SWITCH_RESOURCE_
LIST()
PRODUCT_FIPS140_SSL_SWITCH_RESOURCE_LIST()
PRODUCT_FIPS140_SWITCH_RESOURCE_LIST()
PRODUCT_LIBRARY_FREE()
PRODUCT_LIBRARY_INFO()
PRODUCT_LIBRARY_INFO_TYPE_FROM_STRING()
PRODUCT_LIBRARY_INFO_TYPE_TO_STRING()
PRODUCT_LIBRARY_NEW()
PRODUCT_LIBRARY_VERSION()
PRODUCT_NON_FIPS140_SWITCH_RESOURCE_LIST()
R_CR_asym_decrypt()
R_CR_asym_decrypt_init()
R_CR_asym_encrypt()
R_CR_asym_encrypt_init()
R_CR_CTX_alg_supported()
R_CR_CTX_free()
R_CR_CTX_get_info()
R_CR_CTX_ids_from_sig_id()
R_CR_CTX_ids_to_sig_id()
R_CR_CTX_new()
R_CR_CTX_set_info()
R_CR_decrypt()
R_CR_decrypt_final()
R_CR_decrypt_init()
R_CR_decrypt_update()
R_CR_DEFINE_CUSTOM_CIPHER_LIST()
R_CR_DEFINE_CUSTOM_METHOD_TABLE()
R_CR_derive_key()
R_CR_derive_key_data()
R_CR_digest()
R_CR_digest_final()
R_CR_digest_init()
R_CR_digest_update()
R_CR_dup()
R_CR_encrypt()
R_CR_encrypt_final()
```

```
R_CR_encrypt_init()
R_CR_encrypt_update()
R_CR_free()
R_CR_generate_key()
R_CR_generate_key_init()
R_CR_generate_parameter()
R_CR_generate_parameter_init()
R_CR_get_crypto_provider_name()
R_CR_get_default_imp_method()
R_CR_get_default_method()
R_CR_get_default_signature_map()
R_CR_get_detail()
R_CR_get_detail_string()
R_CR_get_detail_string_table()
R_CR_get_device_handle()
R_CR_get_error()
R_CR_get_error_string()
R_CR_get_file()
R_CR_get_function()
R_CR_get_function_string()
R_CR_get_function_string_table()
R_CR_get_info()
R_CR_get_line()
R_CR_get_reason()
R_CR_get_reason_string()
R_CR_get_reason_string_table()
R_CR_ID_from_string()
R_CR_ID_sign_to_string()
R_CR_ID_to_string()
R_CR_key_exchange_init()
R_CR_key_exchange_phase_1()
R_CR_key_exchange_phase_2()
R_CR_mac()
R_CR_mac_final()
R_CR_mac_init()
R_CR_mac_update()
R_CR_new()
R_CR_random_bytes()
R_CR_random_seed()
R_CR_RES_CRYPTO_CUSTOM_METHOD()
R_CR_set_info()
R_CR_sign()
R_CR_sign_final()
R_CR_sign_init()
R_CR_sign_update()
R_CR_SUB_from_string()
R_CR_SUB_to_string()
R_CR_TYPE_from_string()
R_CR_TYPE_to_string()
R_CR_verify()
R_CR_verify_final()
R_CR_verify_init()
R_CR_verify_mac()
R_CR_verify_mac_final()
R_CR_verify_mac_init()
R_CR_verify_mac_update()
R_CR_verify_update()
R_ERROR_EXIT_CODE()
R_FIPS140_free()
R_FIPS140_get_default()
R_FIPS140_get_failure_reason()
R_FIPS140_get_failure_reason_string()
R_FIPS140_get_info()
R_FIPS140_get_interface_version()
R_FIPS140_get_mode()
R_FIPS140_get_role()
R_FIPS140_get_supported_interfaces()
R_FIPS140_KAT_STATE_to_string()
R_FIPS140_library_free()
R_FIPS140_library_init()
R_FIPS140_load_module()
R_FIPS140_MODE_from_string()
R_FIPS140_MODE_to_string()
R_FIPS140_new()
R_FIPS140_RESULT_from_string()
R_FIPS140_RESULT_to_string()
R_FIPS140_ROLE_from_string()
R_FIPS140_ROLE_to_string()
R_FIPS140_self_tests_full()
R_FIPS140_self_tests_short()
R_FIPS140_set_info()
R_FIPS140_set_interface_version()
R_FIPS140_set_mode()
R_FIPS140_set_role()
R_FIPS140_STATE_from_string()
R_FIPS140_STATE_to_string()
R_FIPS140_unload_module()
R_FORMAT_from_string()
R_FORMAT_to_string()
R_free()
R_get_mem_functions()
R_HW_CTX_build_device_handle_list()
R_HW_CTX_free()
R_HW_CTX_get_device_handle_list()
R_HW_CTX_get_device_handle_list_count()
R_HW_CTX_get_device_handle_list_handle()
R_HW_CTX_get_info()
R_HW_CTX_iterate_devices()
R_HW_CTX_new()
R_HW_CTX_probe_devices()
R_HW_CTX_set_info()
R_HW_DEV_get_device_driver_id()
R_HW_DEV_get_device_name()
R_HW_DEV_get_device_number()
R_HW_DEV_get_info()
R_HW_DEV_is_equal()
R_HW_DEV_set_info()
R_HW_DRIVER_free()
R_HW_DRIVER_get_info()
R_HW_DRIVER_load_devices()
R_HW_DRIVER_new()
R_HW_DRIVER_probe_devices()
R_HW_DRIVER_set_info()
R_HW_OBJ_dup()
R_HW_OBJ_free()
R_HW_OBJ_get_info()
R_HW_OBJ_init()
R_HW_OBJ_new()
R_HW_OBJ_set_info()
R_HW_SEARCH_eof()
R_HW_SEARCH_free()
R_HW_SEARCH_get_locate_count()
```

```
R_HW_SEARCH_locate()
R_HW_SEARCH_new()
R_HW_SEARCH_next()
R_HW_SEARCH_set_browse()
R_LIB_CTX_free()
R_LIB_CTX_get_detail_string()
R_LIB_CTX_get_error_string()
R_LIB_CTX_get_function_string()
R_LIB_CTX_get_info()
R_LIB_CTX_get_reason_string()
R_LIB_CTX_new()
R_LIB_CTX_set_info()
R_lock_ctrl()
R_lock_get_cb()
R_lock_get_name()
R_lock_num()
R_lock_r()
R_lock_w()
R_lock_set_cb()
R_locked_add()
R_locked_add_get_cb()
R_locked_add_set_cb()
R_lockid_new()
R_lockids_free()
R_malloc()
R_PKEY_cmp()
R_PKEY_CTX_free()
R_PKEY_CTX_get_info()
R_PKEY_CTX_get_LIB_CTX()
R_PKEY_CTX_new()
R_PKEY_CTX_set_info()
R_PKEY_decode_pkcs8()
R_PKEY_delete_device()
R_PKEY_encode_pkcs8()
R_PKEY_FORMAT_from_string()
R_PKEY_FORMAT_to_string()
R_PKEY_free()
R_PKEY_from_binary()
R_PKEY_from_bio()
R_PKEY_from_file()
R_PKEY_from_public_key_binary()
R_PKEY_get_handle()
R_PKEY_get_info()
R_PKEY_get_num_bits()
R_PKEY_get_num_primes()
R_PKEY_get_PKEY_CTX()
R_PKEY_get_private_handle()
R_PKEY_get_public_handle()
R_PKEY_get_purpose()
R_PKEY_get_type()
R_PKEY_iterate_fields()
R_PKEY_METHOD_free()
R_PKEY_METHOD_get_flag()
R_PKEY_METHOD_get_name()
R_PKEY_METHOD_get_type()
R_PKEY_new()
R_PKEY_PASSWORD_TYPE_from_string()
R_PKEY_PASSWORD_TYPE_to_string()
R_PKEY_pk_method()
R_PKEY_print()
R_PKEY_public_cmp()
R_PKEY_public_from_bio()
R_PKEY_public_from_file()
R_PKEY_public_to_bio()
R_PKEY_public_to_file()
R_PKEY_read_device()
R_PKEY_reference_inc()
R_PKEY_rsa_blinding_lib_start()
R_PKEY_rsa_no_blinding_lib_start()
R_PKEY_set_handle()
R_PKEY_set_info()
R_PKEY_set_private_handle()
R_PKEY_set_public_handle()
R_PKEY_set_purpose()
R_PKEY_to_binary()
R_PKEY_to_bio()
R_PKEY_to_file()
R_PKEY_to_public_key_binary()
R_PKEY_TYPE_from_string()
R_PKEY_TYPE_to_string()
R_PKEY_write_device()
R_realloc()
R_remalloc()
R_RES_LIST_get_item()
R_RES_LIST_get_resource()
R_RES_LIST_set_item()
R_RES_LIST_set_resource()
R_set_mem_functions()
R_SKEY_delete_device()
R_SKEY_free()
R_SKEY_get_handle()
R_SKEY_get_info()
R_SKEY_new()
R_SKEY_read_device()
R_SKEY_set_handle()
R_SKEY_set_info()
R_SKEY_write_device()
R_thread_id()
R_thread_id_get_cb()
R_thread_id_set_cb()
R_TIME_cmp()
R_time_cmp()
R_TIME_CTX_free()
R_TIME_CTX_new()
R_TIME_dup()
R_TIME_export()
R_TIME_free()
R_TIME_get_time_mi_method()
R_TIME_get_utc_time_method()
R_TIME_import()
R_TIME_new()
R_TIME_offset()
R_TIME_time()
R_unlock_r()
R_unlock_w()
```

4 Acronyms

The following table lists and describes the acronyms and definitions used throughout this document.

Table 7 Acronyms and Definitions

Term	Definition
AES	Advanced Encryption Standard. A fast block cipher with a 128-bit block, and keys of lengths 128, 192, and 256 bits. Replaces DES as the US symmetric encryption standard.
API	Application Programming Interface.
Attack	Either a successful or unsuccessful attempt at breaking part or all of a cryptosystem. Various attack types include an algebraic attack, birthday attack, brute force attack, chosen ciphertext attack, chosen plaintext attack, differential cryptanalysis, known plaintext attack, linear cryptanalysis, and middle person attack.
CBC	Cipher Block Chaining. A mode of encryption in which each ciphertext depends upon all previous ciphertexts. Changing the Initialization Vector (IV) alters the ciphertext produced by successive encryptions of an identical plaintext.
CFB	Cipher Feedback. A mode of encryption that produces a stream of ciphertext bits rather than a succession of blocks. In other respects, it has similar properties to the CBC mode of operation.
CRNG	Continuous Random Number Generation.
CTR	Counter mode of encryption that turns a block cipher into a stream cipher. It generates the next keystream block by encrypting successive values of a counter.
DES	Data Encryption Standard. A symmetric encryption algorithm with a 56-bit key. See also Triple-DES.
Diffie-Hellman	The Diffie-Hellman asymmetric key exchange algorithm. There are many variants, but typically two entities exchange some public information (for example, public keys or random values) and combines them with their own private keys to generate a shared session key. As private keys are not transmitted, eavesdroppers are not privy to all of the information that composes the session key.
DSA	Digital Signature Algorithm. An asymmetric algorithm for creating digital signatures.
DRBG	Deterministic Random Bit Generator.
Dual ECDRBG	Dual Elliptic Curve Deterministic Random Bit Generator.
EC	Elliptic Curve.
ECAES	Elliptic Curve Asymmetric Encryption Scheme.
ECB	Electronic Codebook. A mode of encryption that divides a message into blocks and encrypts each block separately.
ECC	Elliptic Curve Cryptography.

Table 7 Acronyms and Definitions

Term	Definition
ECDH	Elliptic Curve Diffie-Hellman.
ECDHC	Elliptic Curve Diffie-Hellman with Components. Described NIST SP 800-56A, March 2007, Section 5.7.1.2 Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) Primitive.
ECDSA	Elliptic Curve Digital Signature Algorithm.
ECIES	Elliptic Curve Integrated Encryption Scheme.
Encryption	The transformation of plaintext into an apparently less readable form (called ciphertext) through a mathematical process. The ciphertext can be read by anyone who has the key that decrypts (undoes the encryption) the ciphertext.
FIPS	Federal Information Processing Standards.
GCM	Galois/Counter Mode. A mode of encryption that combines the Counter mode of encryption with Galois field multiplication for authentication.
GMAC	Galois Message Authentication Code. An authentication only variant of GCM.
HMAC	Keyed-Hashing for Message Authentication Code.
HMAC DRBG	HMAC Deterministic Random Bit Generator.
IV	Initialization Vector. Used as a seed value for an encryption operation.
KAT	Known Answer Test.
Key	A string of bits used in cryptography, allowing people to encrypt and decrypt data. Can be used to perform other mathematical operations as well. Given a cipher, a key determines the mapping of the plaintext to the ciphertext. The types of keys include distributed key, private key, public key, secret key, session key, shared key, subkey, symmetric key, and weak key.
MD5	A secure hash algorithm created by Ron Rivest. MD5 hashes an arbitrary-length input into a 16-byte digest.
NIST	National Institute of Standards and Technology. A division of the US Department of Commerce (formerly known as the NBS) which produces security and cryptography-related standards.
OFB	Output Feedback. A mode of encryption in which the cipher is decoupled from its ciphertext.
OS	Operating System.
PC	Personal Computer.
PDA	Personal Digital Assistant.
PPC	PowerPC.
privacy	The state or quality of being secluded from the view or presence of others.

Table 7 Acronyms and Definitions

Term	Definition
private key	The secret key in public key cryptography. Primarily used for decryption but also used for encryption with digital signatures.
PRNG	Pseudo-random Number Generator.
RC2	Block cipher developed by Ron Rivest as an alternative to the DES. It has a block size of 64 bits and a variable key size. It is a legacy cipher and RC5 should be used in preference.
RC4	Symmetric algorithm designed by Ron Rivest using variable length keys (usually 40-bit or 128-bit).
RC5	Block cipher designed by Ron Rivest. It is parameterizable in its word size, key length, and number of rounds. Typical use involves a block size of 64 bits, a key size of 128 bits, and either 16 or 20 iterations of its round function.
RNG	Random Number Generator.
RSA	Public key (asymmetric) algorithm providing the ability to encrypt data and create and verify digital signatures. RSA stands for Rivest, Shamir, and Adleman, the developers of the RSA public key cryptosystem.
SHA	Secure Hash Algorithm. An algorithm that creates a unique hash value for each possible input. SHA takes an arbitrary input that is hashed into a 160-bit digest.
SHA-1	A revision to SHA to correct a weakness. It produces 160-bit digests. SHA-1 takes an arbitrary input that is hashed into a 20-byte digest.
SHA-2	The NIST-mandated successor to SHA-1, to complement the Advanced Encryption Standard. It is a family of hash algorithms (SHA-224, SHA-256, SHA-384 and SHA-512) that produce digests of 224, 256, 384 and 512 bits respectively.
Triple-DES	A variant of DES that uses three 56-bit keys.