RSA BSAFE® Crypto-C Micro Edition
Version 4.0.1
Security Policy (SPARC T4) Level 1

This is a non-proprietary Security Policy (for SPARC T4) for RSA BSAFE Crypto-C Micro Edition 4.0.1 (Crypto-C ME). It describes how Crypto-C ME meets the Level 1 security requirements of FIPS 140-2, the Level 3 security requirements of FIPS 140-2 for design assurance, and how to securely operate Crypto-C ME in a FIPS 140-2-compliant manner. This Security Policy is prepared as part of the FIPS 140-2 Level 1 validation of Crypto-C ME.

FIPS 140-2 (Federal Information Processing Standards Publication 140-2 - *Security Requirements for Cryptographic Modules*) details the United States Government requirements for cryptographic modules. For more information about the FIPS 140-2 standard and validation program, go to the NIST Web site at http://csrc.nist.gov/cryptval/.

This document may be freely reproduced and distributed whole and intact including the Copyright Notice.

Contents:

1 Introduction

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.

Note: In this document, the term *cryptographic module*, refers to the Crypto-C ME FIPS 140-2 Level 1 validated cryptographic module.

1.1 References

This document deals only with the operations and capabilities of the Crypto-C ME cryptographic module in terms of a FIPS 140-2 cryptographic module security policy. For more information about Crypto-C ME 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/`.

1.2 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, "Introduction" on page 2 provides an overview and introduction to the Security Policy.

- "Crypto-C ME Cryptographic Toolkit" on page 3 describes Crypto-C ME and how it meets FIPS 140-2 requirements.

- "Secure Operation of Crypto-C ME" on page 15 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 and Definitions" on page 25 lists the acronyms and definitions used in this document.

With the exception of the non-proprietary Security Policy documents, the FIPS 140-2 validation submission documentation is EMC Corporation-proprietary and is releasable only under appropriate non-disclosure agreements. For access to these documents, please contact RSA.

2 Crypto-C ME Cryptographic Toolkit

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 available algorithms cannot be changed.

2.1 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 is validated as meeting all FIPS 140-2 Level 1 security 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.

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 Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	3
Mitigation of Other Attacks	1
Overall	1

2.1.1 Laboratory Validated Operating Environments

For FIPS 140-2 validation, Crypto-C ME is tested by an accredited FIPS 140-2 testing laboratory on Oracle® Solaris® 10, SPARC T4, part number 527-1437-01. Compliance is maintained in all operating environments for which the binary executable remains unchanged.

The CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when so ported if the specific operational environment is not listed on the validation certificate.

2.1.2 Configuring Single User Mode

To configure single user mode for systems running a Oracle Solaris 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. Edit `/etc/inet/inetd.conf` to remove or comment out the lines for remote login, remote command execution, and file transfer daemons.

5. Reboot the system for the changes to take effect.

2.2 Crypto-C ME Interfaces

Crypto-C ME is validated as a multi-chip standalone cryptographic 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 master and resource shared library files, and signature files that comprise the module:

- Master shared library: `libcryptocme.so`

- Resource shared libraries: `libccme_base.so`, `libccme_base_non_fips.so`, `libccme_asym.so`, `libccme_ecdrbg.so`, `libccme_ecc.so`, `libccme_ecc_non_fips.so`, `libccme_ecc_accel_fips.so`, `libccme_ecc_accel_non_fips.so`, and `libccme_error_info.so`

- Signature files: `cryptocme.sig` and `cryptocme_test_on_use.sig`.

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
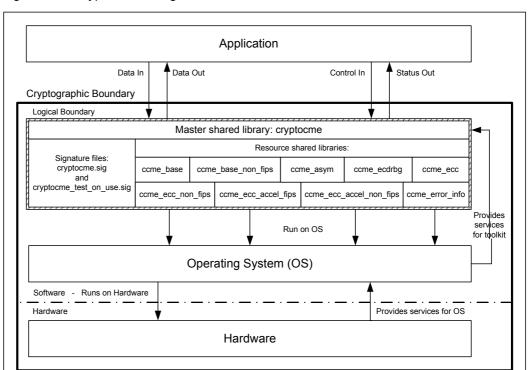and error codes that are documented for each call. This is illustrated in the following
diagram.

Figure 1   Crypto-C ME Logical Interfaces

2.3 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 Crypto 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.

2.3.1 Crypto Officer Role

The Crypto Officer is responsible for installing and loading the cryptographic module. After the module is installed and operational, an operator can assume the Crypto Officer role by calling `R_PROV_FIPS140_assume_role()` with `R_FIPS140_ROLE_OFFICER`. An operator assuming the Crypto Officer role can call any Crypto-C ME function. For a complete list of functions available to the Crypto Officer, see "Services" on page 19.

2.3.2 Crypto User Role

An operator can assume the Crypto User role by calling `R_PROV_FIPS140_assume_role()` with `R_FIPS140_ROLE_USER`. An operator assuming the Crypto User role can use the entire Crypto-C ME API except for `R_PROV_FIPS140_self_test_full()`, which is reserved for the Crypto Officer. For a complete list of Crypto-C ME functions, see "Services" on page 19.

2.4 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.

2.4.1 Key Generation

Crypto-C ME supports generation of RSA, Diffie-Hellman (DH) and Elliptic Curve Cryptography (ECC) public and private keys. Also, Crypto-C ME uses an HMAC Deterministic Random Bit Generator (DRBG) in the generation of asymmetric and symmetric keys used in algorithms such as AES, Triple-DES, RSA, Diffie-Hellman, ECC, and HMAC.

2.4.2 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 2   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).
Diffie-Hellman 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).
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).

2.4.3 Key Access

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

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

The following table lists the different services provided by the toolkit with the type of access to keys or CSPs.

Table 3   Key and CSP Access

Service	Key or CSP	Type of Access
Encryption and decryption	Symmetric keys (AES, Triple-DES)	Read/Execute
Digital signature and verification	Asymmetric keys (RSA, DSA - verification only, and ECDSA - verification only)	Read/Execute
Hashing	None	N/A
MAC	HMAC keys	Read/Execute
Random number generation	HMAC DRBG entropy, V, key, and init_seed	Read/Write/Execute
Key generation	Symmetric keys (AES, Triple-DES) Asymmetric keys (RSA, ECDSA, DH, ECDH) MAC keys (HMAC)	Write
Key establishment primitives	Asymmetric keys (RSA, DH, ECDH)	Read/Execute
Self-test (Crypto Officer service)	Hardcoded keys (DSA and AES)	Read/Execute
Show status	None	N/A
Zeroization	All	Read/Write

2.4.4 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 Developers Guide* to ensure sensitive data is protected by zeroizing the data from memory when it is no longer needed.

2.5 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.

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

Algorithm	Validation Certificate
AES CBC, CFB128, ECB, OFB, CTR, and CCM (with 128, 192, and 256-bit key sizes)	2017
AES XTS (with 128 and 256-bit key sizes)	2017
AES GCM with automatic Initialization Vector (IV) generation (with 128, 192, and 256-bit key sizes). For more information, see Chapter 5, Cryptographic Operations in the *RSA BSAFE Crypto-C Micro Edition Developers Guide*.	2017
Triple-DES ECB, CBC, CFB (64-bit), and OFB (64-bit).	1302
Diffie-Hellman (2048 to 4096-bit key size) and Elliptic Curve Diffie-Hellman (224 to 571-bit key size)	Non-approved (Allowed in FIPS 140-2 mode).
DSA (signature verification only)	642
ECDSA (signature verification only)	292
HMAC DRBG	191
RSA X9.31, PKCS#1 V.1.5, and PKCS#1 V.2.1 (SHA256 - PSS)	1046
RSA encrypt and decrypt (2048 to 4096-bit key size). For key wrapping using RSA, the key establishment methodology provides between 112 and 150 bits of encryption strength. Less than 112 bits of encryption strength (key sizes less than 2048 bits) is non-compliant.	Non-approved (Allowed in FIPS 140-2 mode for key transport).
SHA-1	1767
SHA-224, 256, 384, and 512	1767
HMAC-SHA1, SHA224, SHA256, SHA384, and SHA512	1221

The following algorithms are not FIPS 140-2-approved:

- DES
- Camellia
- MD2
- MD4
- MD5
- HMAC MD5
- DES40
- RC2
- RC4
- RC5
- RSA with key sizes less than 2048 bits
- DSA for signature generation
- ECDSA for signature generation
- DH with key sizes less than 2048 bits
- ECDH with key sizes less than 224 bits
- ECAES
- ECIES
- PBKDF1 SHA-1
- PBKDF2 HMAC SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512
- Dual EC DRBG
- Non-deterministic Random Number Generator (NDRNG) (Entropy)
- Non-approved RNG (FIPS 186-2)
- Non-approved RNG (OTP).

For more information about using Crypto-C ME in a FIPS 140-2-compliant manner, see "Secure Operation of Crypto-C ME" on page 15.

2.6 Hardware Instructions

Hardware instructions provided by the Oracle SPARC T4 processor are utilized by Crypto-C ME to optimize cryptographic performance. The following table lists and describes the hardware instructions used.

Table 5   Oracle SPARC T4 Hardware Instructions used by Crypto-C ME

Instruction	Description
AES_EROUND01	AES encryption operation, on columns 0 and 1
AES_EROUND23	AES encryption operation, on columns 2 and 3
AES_DROUND01	AES decryption operation, on columns 0 and 1
AES_DROUND23	AES decryption operation, on columns 2 and 3
AES_EROUND01_LAST	The last round of AES encryption, on columns 0 and 1
AES_EROUND23_LAST	The last round of AES encryption, on columns 2 and 3
AES_DROUND01_LAST	The last round of AES decryption, on columns 0 and 1
AES_DROUND23_LAST	The last round of AES decryption, on columns 2 and 3
AES_KEXPAND1	AES key expansion operation, with RCON
AES_KEXPAND0	AES key expansion operation, without RCON
AES_KEXPAND2	AES key expansion operation, without SBOX
DES_ROUND	DES encryption operation
DES_IP	DES Initial Permutation function
DES_IIP	DES Inverse Initial Permutation function
DES_KEXPAND	DES key expansion operation
MD5	MD5 message digest operation
SHA1	SHA1 message digest operation
SHA256	SHA256 message digest operation
SHA512	SHA512 message digest operation
MPMUL	Multiple Precision Multiply operation
MONTMUL	Montgomery Multiply operation
MONTSQR	Montgomery Squaring operation
XMULX	Bitwise multiply on the low 64 bits
XMULXHI	Bitwise multiply on the high 64 bits

2.7 Self Tests

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

If a power-up self-test fails for one of the resource libraries, all cryptographic services for that library are disabled. Services for a disabled library can only be re-enabled by reloading the FIPS 140-2 module. If a conditional self-test fails, the operation fails but no services are disabled.

2.7.1 Power-up Self-test

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

*   AES, AES CCM, AES GCM, AES GMAC, and AES XTS Known Answer Tests (KATs)

*   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

*   Diffie-Hellman and Elliptic Curve Diffie-Hellman conditional tests

*   ECDSA sign/verify test

*   HMAC DRBG KATs

*   Software integrity test using DSA signature verification.

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

2.7.2 Conditional Self-tests

Crypto-C ME performs two conditional self-tests:

*   A pair-wise consistency test each time Crypto-C ME generates an 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 140-2 standard. The CRNG test is performed on DRNGs (FIPS 186-2 PRNG - Change Notice 1, with and without the mod q step; Dual EC DRBG; HMAC DRBG; NDRNG (Entropy); Non-Approved RNG (OTP).

2.7.3 Critical Functions Tests

Crypto-C ME performs known answer tests for:

- MD5 and HMAC-MD5, which are available in modes
  `R_LIB_CTX_MODE_FIPS140_SSL` and `R_LIB_CTX_MODE_JCMVP_SSL`.

- Camellia ECB, CBC, CFB, and OFB for key sizes 128, 192, and 256 bits, which
  are available in modes `R_LIB_CTX_MODE_JCMVP` and
  `R_LIB_CTX_MODE_JCMVP_SSL`.

2.7.4 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.

3 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.

3.1 Crypto Officer and Crypto User Guidance

The Crypto Officer and Crypto 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:

- Bit lengths for an RSA key pair must be between 2048 and 4096 bits in multiples of 512.

- Bit lengths for an HMAC key must be between 112 and 4096 bits.

- EC key pairs must have named curve domain parameters from the set of NIST-recommended named curves (P224, P256, P384, P521, B233, B283, B409, B571, K233, K283, K409, K571). Named curves P192, B163, and K163 are allowed for legacy-use.

- When using RSA for key wrapping, the strength of the methodology is between 112 and 150 bits of security.

- The Diffie-Hellman shared secret provides between 112 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 112 and 256 bits of encryption strength.

- You must set the feature, `R_FIPS140_FEATURE_no_default_ecdrbg`, against the FIPS 140-2 provider prior to loading the module to ensure one of the HMAC DRBG implementations is set as the default PRNG.

- When using HMAC DRBG to generate keys, the requested security strength must be at least as great as the security strength of the key being generated.

- When using GCM feedback mode for symmetric encryption, the authentication tag length and authenticated data length may be specified as input parameters, but the Initialization Vector (IV) must not be specified. It must be generated internally.

- In the case where the module is powered down, a new key must be used for AES GCM encryption/decryption.

3.2 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 6   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_PROV_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.

3.3 Modes of Operation

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

Table 7   Crypto-C ME Modes of Operation

Mode Filter	Description
R_LIB_CTX_MODE_STANDARD Not FIPS 140-2-approved.	Allows users to operate Crypto-C ME without any cryptographic algorithm restrictions. This is the Crypto-C ME default mode on startup.
R_LIB_CTX_MODE_FIPS140 FIPS 140-2-approved.	Provides the cryptographic algorithms listed in Table 4 on page 10.
R_LIB_CTX_MODE_FIPS140_SSL FIPS 140-2-approved if used with TLS protocol implementations.	Provides the same algorithms as R_LIB_CTX_MODE_FIPS140, plus the MD5 message digest algorithm. 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.

Table 7   Crypto-C ME Modes of Operation (continued)

Mode Filter	Description
`R_LIB_CTX_MODE_JCMVP` Not FIPS 140-2-approved.	Provides the cryptographic algorithms approved by the Japan Cryptographic Module Validation Program (JCMVP).
`R_LIB_CTX_MODE_JCMVP_SSL` Not FIPS 140-2-approved.	Provides the cryptographic algorithms approved by the JCMVP, plus the MD5 message digest algorithm.

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 Crypto 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.

3.4 Operating Crypto-C ME

Crypto-C ME operates in `R_LIB_CTX_MODE_STANDARD` mode by default on startup. The current Crypto-C ME mode is determined by calling `R_LIB_CTX_get_info()` with `R_LIB_CTX_INFO_ID_MODE`.

To change the module to another mode, call `R_LIB_CTX_set_mode()` with one of the mode identifiers listed in Table 7 on page 16.

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_LIB_CTX_set_mode()` with `NULL` to enable `R_LIB_CTX_MODE_STANDARD`.

`R_PROV_FIPS140_self_tests_full()` is restricted to operation by the Crypto Officer.

The user of Crypto-C ME links with the `ccme_core` and `ccme_fipsprov` static libraries for their platform. At run time, `ccme_fipsprov` loads the `cryptocme` master shared library, which then loads all of the resource shared libraries. For more information, see "FIPS 140-2 Libraries" in Chapter 7, FIPS 140-2 Operations in the *RSA BSAFE Crypto-C Micro Edition Developers Guide*.

The current Crypto-C ME role is determined by calling `R_LIB_CTX_get_info()` with `R_LIB_CTX_INFO_ID_ROLE`. The role is changed by calling `R_PROV_FIPS140_assume_role()` with one of the information identifiers listed in Table 7 on page 16.

3.5 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*.

3.6 Deterministic Random Number Generator

In all modes of operation, Crypto-C ME provides HMAC DRBG as the default deterministic random number generator (DRNG).

Users can choose to use a different HMAC DRBG when creating a cryptographic object and setting this object against the operation requiring random number generation (for example, key generation). When DSA is used, the DRBG used internally is always HMAC DRBG.

Crypto-C ME also includes a non-approved NDRNG (Entropy) used to generate seed material for the DRNGs.

3.6.1 DRNG Seeding

In the FIPS 140-2 validated library, Crypto-C ME implements DRNGs that can be called to generate random data. The quality of the random data output from these DRNGs depend 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.

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, see the *RSA BSAFE Crypto-C Micro Edition API Reference Guide*.

3.7 Physical Security

Crypto-C ME is validated as a multi-chip standalone cryptographic module and is being validated at Level 1 for physical security. The contents of the module, including all hardware, firmware, software, and data (including plaintext cryptographic keys and unprotected CSPs) is protected by the case of the general-purpose computer or mobile device, which encloses the hardware running the module.

4 Services

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

R_add()	BIO_new_file_w()
BIO_append_filename()	BIO_new_file_w_ef()
BIO_cb_cmd_to_string()	BIO_new_fp()
BIO_cb_post()	BIO_new_fp_ef()
BIO_cb_pre()	BIO_new_init()
BIO_CB_return()	BIO_new_init_ef()
BIO_clear_flags()	BIO_new_mem()
BIO_clear_retry_flags()	BIO_new_mem_ef()
BIO_copy_next_retry()	BIO_open_file()
BIO_ctrl()	BIO_open_file_w()
BIO_debug_cb()	BIO_pending()
BIO_dump()	BIO_pop()
BIO_dump_format()	BIO_print_hex()
BIO_dup_chain()	BIO_printf()
BIO_dup_chain_ef()	BIO_push()
BIO_eof()	BIO_puts()
BIO_f_buffer()	BIO_read()
BIO_f_null()	BIO_read_filename()
BIO_find_type()	BIO_reference_inc()
BIO_flags_to_string()	BIO_reset()
BIO_flush()	BIO_retry_type()
BIO_free()	BIO_rw_filename()
BIO_free_all()	BIO_s_file()
BIO_get_app_data()	BIO_s_mem()
BIO_get_buffer_num_lines()	BIO_s_null()
BIO_get_cb()	BIO_seek()
BIO_get_cb_arg()	BIO_set()
BIO_get_close()	BIO_set_app_data()
BIO_get_flags()	BIO_set_bio_cb()
BIO_get_fp()	BIO_set_buffer_read_data()
BIO_get_info_cb()	BIO_set_buffer_size()
BIO_get_mem_data()	BIO_set_cb()
BIO_get_retry_BIO()	BIO_set_cb_arg()
BIO_get_retry_flags()	BIO_set_cb_recursive()
BIO_get_retry_reason()	BIO_set_close()
BIO_get_state_cb()	BIO_set_flags()
BIO_get_state_cb_arg()	BIO_set_fp()
BIO_gets()	BIO_set_info_cb()
BIO_method_name()	BIO_set_mem_eof_return()
BIO_method_type()	BIO_set_read_buffer_size()
BIO_new()	BIO_set_retry_read()
BIO_new_ef()	BIO_set_retry_small_buffer()
BIO_new_file()	BIO_set_retry_special()
BIO_new_file_ef()	BIO_set_retry_write()

```
BIO_set_state_cb()
BIO_set_write_buffer_size()
BIO_should_io_special()
BIO_should_read()
BIO_should_retry()
BIO_should_small_buffer()
BIO_should_write()
BIO_state_to_string()
BIO_tell()
BIO_wpending()
BIO_write()
BIO_write_filename()
R_BASE64_decode()
R_BASE64_decode_checked()
R_BASE64_decode_checked_ef()
R_BASE64_decode_ef()
R_BASE64_encode()
R_BASE64_encode_checked()
R_BASE64_encode_checked_ef()
R_BASE64_encode_ef()
R_BUF_append()
R_BUF_assign()
R_BUF_cmp()
R_BUF_cmp_raw()
R_BUF_consume()
R_BUF_cut()
R_BUF_dup()
R_BUF_free()
R_BUF_get_data()
R_BUF_grow()
R_BUF_insert()
R_BUF_join()
R_BUF_length()
R_BUF_max_length()
R_BUF_new()
R_BUF_prealloc()
R_BUF_reset()
R_BUF_resize()
R_BUF_strdup()
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_new_ef()
R_CR_CTX_reference_inc()
R_CR_CTX_set_info()
R_CR_decrypt()
```

```
R_CR_decrypt_final()
R_CR_decrypt_init()
R_CR_decrypt_update()
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_dup_ef()
R_CR_encrypt()
R_CR_encrypt_final()
R_CR_encrypt_init()
R_CR_encrypt_update()
R_CR_entropy_bytes()
R_CR_export_params()
R_CR_free()
R_CR_generate_key()
R_CR_generate_key_init()
R_CR_generate_parameter()
R_CR_generate_parameter_init()
R_CR_get_detail()
R_CR_get_detail_string()
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_info()
R_CR_get_line()
R_CR_get_reason()
R_CR_get_reason_string()
R_CR_ID_from_string()
R_CR_ID_sign_to_string()
R_CR_ID_to_string()
R_CR_import_params()
R_CR_key_exchange_init()
R_CR_key_exchange_phase_1()
R_CR_key_exchange_phase_2()
R_CR_keywrap_init()
R_CR_keywrap_unwrap()
R_CR_keywrap_unwrap_init()
R_CR_keywrap_unwrap_PKEY()
R_CR_keywrap_unwrap_SKEY()
R_CR_keywrap_wrap()
R_CR_keywrap_wrap_init()
R_CR_keywrap_wrap_PKEY()
R_CR_keywrap_wrap_SKEY()
R_CR_mac()
R_CR_mac_final()
R_CR_mac_init()
R_CR_mac_update()
```

```
R_CR_new()
R_CR_new_ef()
R_CR_next_error()
R_CR_random_bytes()
R_CR_random_init()
R_CR_random_reference_inc()
R_CR_random_seed()
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_validate_parameters()
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()
ERR_STATE_add_error_data()
ERR_STATE_clear_error()
ERR_STATE_error_string()
ERR_STATE_free_strings()
ERR_STATE_func_error_string()
ERR_STATE_get_error()
ERR_STATE_get_error_line()
ERR_STATE_get_error_line_data()
ERR_STATE_get_next_error_library()
ERR_STATE_get_state()
ERR_STATE_lib_error_string()
ERR_STATE_load_ERR_strings()
ERR_STATE_load_strings()
ERR_STATE_peek_error()
ERR_STATE_peek_error_line()
ERR_STATE_peek_error_line_data()
ERR_STATE_peek_last_error()
ERR_STATE_peek_last_error_line()
ERR_STATE_peek_last_error_line_data()
ERR_STATE_print_errors()
ERR_STATE_print_errors_fp()
ERR_STATE_put_error()
ERR_STATE_reason_error_string()
ERR_STATE_remove_state()
ERR_STATE_set_error_data()
R_ERR_STACK_clear_error()
R_ERR_STACK_free()
R_ERR_STACK_get_error_state()
```

```
R_ERR_STACK_new()
R_ERR_STACK_put_error_state()
R_ERR_STATE_free()
R_ERR_STATE_get_error()
R_ERR_STATE_get_error_line()
R_ERR_STATE_get_error_line_data()
R_ERR_STATE_new()
R_ERR_STATE_set_error_data()
R_ERROR_EXIT_CODE()
R_FILTER_sort()
R_FORMAT_from_string()
R_FORMAT_to_string()
R_ITEM_cmp()
R_ITEM_destroy()
R_ITEM_dup()
R_LIB_CTX_add_filter()
R_LIB_CTX_add_provider()
R_LIB_CTX_add_resource()
R_LIB_CTX_add_resources()
R_LIB_CTX_dup()
R_LIB_CTX_dup_ef()
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_new_ef()
R_LIB_CTX_reference_inc()
R_LIB_CTX_set_info()
R_LIB_CTX_set_mode()
R_lock()
R_LOCK_add()
R_lock_ctrl()
R_LOCK_exec()
R_LOCK_free()
R_lock_get_cb()
R_lock_get_name()
R_LOCK_lock()
R_LOCK_new()
R_lock_num()
R_lock_r()
R_lock_set_cb()
R_LOCK_unlock()
R_lock_w()
R_locked_add()
R_locked_add_get_cb()
R_locked_add_set_cb()
R_lockid_new()
R_lockids_free()
R_MEM_clone()
R_MEM_compare()
```

R_MEM_delete()
R_MEM_free()
R_MEM_get_global()
R_MEM_malloc()
R_MEM_new_callback()
R_MEM_new_default()
R_MEM_realloc()
R_MEM_strdup()
R_MEM_zfree()
R_MEM_zmalloc()
R_MEM_zrealloc()
R_os_clear_sys_error()
R_os_get_last_sys_error()
PRODUCT_DEFAULT_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()
R_PAIRS_add()
R_PAIRS_clear()
R_PAIRS_free()
R_PAIRS_generate()
R_PAIRS_get_info()
R_PAIRS_init()
R_PAIRS_init_ef()
R_PAIRS_new()
R_PAIRS_new_ef()
R_PAIRS_next()
R_PAIRS_parse()
R_PAIRS_parse_allow_sep()
R_PAIRS_reset()
R_PAIRS_set_info()
R_passwd_get_cb()
R_passwd_get_passwd()
R_passwd_set_cb()
R_passwd_stdin_cb()
R_PKEY_cmp()
R_PKEY_copy()
R_PKEY_CTX_free()
R_PKEY_CTX_get_info()
R_PKEY_CTX_get_LIB_CTX()
R_PKEY_CTX_get_memory()
R_PKEY_CTX_new()
R_PKEY_CTX_new_ef()
R_PKEY_CTX_reference_inc()
R_PKEY_CTX_set_info()
R_PKEY_decode_pkcs8()
R_PKEY_delete()
R_PKEY_dup()
R_PKEY_dup_ef()

R_PKEY_EC_NAMED_CURVE_from_string()
R_PKEY_EC_NAMED_CURVE_to_string()
R_PKEY_encode_pkcs8()
R_PKEY_FORMAT_from_string()
R_PKEY_FORMAT_to_string()
R_PKEY_free()
R_PKEY_from_binary()
R_PKEY_from_binary_ef()
R_PKEY_from_bio()
R_PKEY_from_bio_ef()
R_PKEY_from_file()
R_PKEY_from_file_ef()
R_PKEY_from_public_key_binary()
R_PKEY_from_public_key_binary_ef()
R_PKEY_get_info()
R_PKEY_get_num_bits()
R_PKEY_get_num_primes()
R_PKEY_get_PEM_header()
R_PKEY_get_PKEY_CTX()
R_PKEY_get_type()
R_PKEY_is_matching_public_key()
R_PKEY_iterate_fields()
R_PKEY_load()
R_PKEY_new()
R_PKEY_new_ef()
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_bio_ef()
R_PKEY_public_from_file()
R_PKEY_public_from_file_ef()
R_PKEY_public_get_PEM_header()
R_PKEY_public_to_bio()
R_PKEY_public_to_file()
R_PKEY_reference_inc()
R_PKEY_RES_CUSTOM()
R_PKEY_SEARCH_add_filter()
R_PKEY_SEARCH_free()
R_PKEY_SEARCH_init()
R_PKEY_SEARCH_new()
R_PKEY_SEARCH_next()
R_PKEY_set_info()
R_PKEY_set_provider_filter()
R_PKEY_signhash()
R_PKEY_store()
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_public_to_PEM_header()
R_PKEY_TYPE_to_PEM_header()
R_PKEY_TYPE_to_string()
R_PKEY_verifyhash()
R_PROV_ctrl()
R_PROV_FIPS140_assume_role()
R_PROV_FIPS140_authenticate_role()
R_PROV_FIPS140_authenticate_role_with_
token()
R_PROV_FIPS140_free()
R_PROV_FIPS140_get_default_resource_
list()
R_PROV_FIPS140_get_info()
R_PROV_FIPS140_init_roles()
R_PROV_FIPS140_load()
R_PROV_FIPS140_load_env()
R_PROV_FIPS140_new()
R_PROV_FIPS140_reason_string()
R_PROV_FIPS140_self_tests_full()
R_PROV_FIPS140_self_tests_short()
R_PROV_FIPS140_set_info()
R_PROV_FIPS140_set_path()
R_PROV_FIPS140_set_path_w()
R_PROV_FIPS140_set_pin()
R_PROV_FIPS140_set_pin_with_token()
R_PROV_FIPS140_set_roles_file()
R_PROV_FIPS140_set_roles_file_w()
R_PROV_free()
R_PROV_get_default_resource_list()
R_PROV_get_info()
R_PROV_PKCS11_clear_quirks()
R_PROV_PKCS11_close_token_sessions()
R_PROV_PKCS11_get_cryptoki_version()
R_PROV_PKCS11_get_description()
R_PROV_PKCS11_get_driver_name()
R_PROV_PKCS11_get_driver_path()
R_PROV_PKCS11_get_driver_version()
R_PROV_PKCS11_get_flags()
R_PROV_PKCS11_get_info()
R_PROV_PKCS11_get_manufacturer_id()
R_PROV_PKCS11_get_quirks()
R_PROV_PKCS11_get_slot_count()
R_PROV_PKCS11_get_slot_description()
R_PROV_PKCS11_get_slot_firmware_
version()
R_PROV_PKCS11_get_slot_flags()
R_PROV_PKCS11_get_slot_hardware_
version()
R_PROV_PKCS11_get_slot_ids()
R_PROV_PKCS11_get_slot_info()
R_PROV_PKCS11_get_slot_manufacturer_
id()
R_PROV_PKCS11_get_token_default_pin()

R_PROV_PKCS11_get_token_flags()
R_PROV_PKCS11_get_token_info()
R_PROV_PKCS11_get_token_label()
R_PROV_PKCS11_get_token_login_pin()
R_PROV_PKCS11_get_token_manufacturer_
id()
R_PROV_PKCS11_get_token_model()
R_PROV_PKCS11_get_token_serial_
number()
R_PROV_PKCS11_init_token()
R_PROV_PKCS11_init_user_pin()
R_PROV_PKCS11_load()
R_PROV_PKCS11_new()
R_PROV_PKCS11_set_driver_name()
R_PROV_PKCS11_set_driver_path()
R_PROV_PKCS11_set_driver_path_w()
R_PROV_PKCS11_set_info()
R_PROV_PKCS11_set_login_cb()
R_PROV_PKCS11_set_quirks()
R_PROV_PKCS11_set_slot_info()
R_PROV_PKCS11_set_token_login_pin()
R_PROV_PKCS11_set_user_pin()
R_PROV_PKCS11_unload()
R_PROV_PKCS11_update_full()
R_PROV_PKCS11_update_only()
R_PROV_reference_inc()
R_PROV_set_info()
R_PROV_setup_features()
R_PROV_SOFTWARE_add_resources()
R_PROV_SOFTWARE_get_default_resource_
list()
R_PROV_SOFTWARE_new()
R_PROV_SOFTWARE_new_default()
R_RW_LOCK_free()
R_RW_LOCK_new()
R_RW_LOCK_read()
R_RW_LOCK_read_exec()
R_RW_LOCK_unlock()
R_RW_LOCK_write()
R_RW_LOCK_write_exec()
R_SELECT_ctrl()
R_SELECT_dup()
R_SELECT_free()
R_SELECT_get_info()
R_SELECT_set_info()
R_SKEY_delete()
R_SKEY_dup()
R_SKEY_dup_ef()
R_SKEY_free()
R_SKEY_get_info()
R_SKEY_load()
R_SKEY_new()
R_SKEY_new_ef()

R_SKEY_SEARCH_add_filter()
R_SKEY_SEARCH_free()
R_SKEY_SEARCH_init()
R_SKEY_SEARCH_new()
R_SKEY_SEARCH_next()
R_SKEY_set_info()
R_SKEY_set_provider_filter()
R_SKEY_store()
R_STATE_cleanup()
R_STATE_init()
R_STATE_init_defaults()
R_STATE_init_defaults_mt()
R_SYNC_get_method()
R_SYNC_METH_default()
R_SYNC_METH_pthread()
R_SYNC_METH_solaris()
R_SYNC_METH_vxworks()
R_SYNC_METH_windows()
R_SYNC_set_method()
STACK_cat()
STACK_clear()
STACK_clear_arg()
STACK_delete()
STACK_delete_all()
STACK_delete_all_arg()
STACK_delete_ptr()
STACK_dup()
STACK_dup_ef()
STACK_find()
STACK_for_each()
STACK_insert()
STACK_lfind()
STACK_move()
STACK_new()
STACK_new_ef()
STACK_pop()
STACK_pop_free()
STACK_pop_free_arg()
STACK_push()
STACK_set()
STACK_set_cmp_func()
STACK_shift()
STACK_unshift()
STACK_zero()
R_THREAD_create()
R_thread_id()
R_THREAD_id()
R_thread_id_get_cb()
R_thread_id_set_cb()
R_THREAD_init()
R_THREAD_self()
R_THREAD_wait()
R_THREAD_yield()

R_time()
R_TIME_cmp()
R_time_cmp()
R_TIME_CTX_free()
R_TIME_CTX_new()
R_TIME_CTX_new_ef()
R_TIME_dup()
R_TIME_dup_ef()
R_time_export()
R_TIME_export()
R_TIME_export_timestamp()
R_time_free()
R_TIME_free()
R_time_from_int()
R_time_get_cmp_func()
R_time_get_export_func()
R_time_get_func()
R_time_get_import_func()
R_time_get_offset_func()
R_TIME_get_utc_time_method()
R_TIME_import()
R_time_import()
R_TIME_import_timestamp()
R_TIME_new()
R_time_new()
R_TIME_new_ef()
R_time_new_ef()
R_time_offset()
R_TIME_offset()
R_time_set_cmp_func()
R_time_set_export_func()
R_time_set_func()
R_time_set_import_func()
R_time_set_offset_func()
R_time_size()
R_TIME_time()
R_time_to_int()
R_unlock()
R_unlock_r()
R_unlock_w()

5 Acronyms and Definitions

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

Table 8   Acronyms and Definitions

Term	Definition
AES	Advanced Encryption Standard. A fast symmetric key algorithm 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.
Camellia	A symmetric key algorithm with a 128-bit block, and keys of lengths 128, 192, and 256 bits. Developed jointly by Mitsubishi and NTT.
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.
CTS	Cipher text stealing mode of encryption that enables block ciphers to be used to process data that is not evenly divisible into blocks, without the length of the ciphertext increasing.
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.
EC	Elliptic Curve.
ECAES	Elliptic Curve Asymmetric Encryption Scheme.

Table 8   Acronyms and Definitions

Term	Definition
ECB	Electronic Codebook. A mode of encryption that divides a message into blocks and encrypts each block separately.
ECC	Elliptic Curve Cryptography.
ECDH	Elliptic Curve Diffie-Hellman.
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.
JCMVP	Japan Cryptographic Module Validation Program.
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.
MD2	A message digest algorithm that hashes an arbitrary-length input into a 16-byte digest. MD2 is no longer considered secure.
MD4	A message digest algorithm that hashes an arbitrary-length input into a 16-byte digest.
MD5	A message digest algorithm that hashes an arbitrary-length input into a 16-byte digest. Designed as a replacement for MD4.
NDRNG	Non-deterministic random number generator.
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.

Table 8   Acronyms and Definitions

Term	Definition
OS	Operating System.
PBKDF1	Password-based Key Derivation Function 1. A method of password-based key derivation that applies a message digest (MD2, MD5, or SHA-1) to derive the key. PBKDF1 is not recommended for new applications because the message digest algorithms used have known vulnerabilities, and the derived keys are limited in length.
PBKDF2	Password-based Key Derivation Function 2. A method of password-based key derivation that applies a Message Authentication Code (MAC) algorithm to derive the key.
PC	Personal Computer.
PDA	Personal Digital Assistant.
PPC	PowerPC.
privacy	The state or quality of being secluded from the view or presence of others.
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.
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.
XTS	XEX-based Tweaked Codebook mode with ciphertext stealing. A mode of encryption used with AES.