Dell EMC Data Domain Crypto-C Micro Edition

Version 4.0.1

Security Policy Level 1 with Level 2 Roles, Services, and Authentication

This is a non-proprietary Security Policy for Dell EMC Data Domain Crypto-C Micro Edition 4.0.1 (Crypto-C ME). It describes how Crypto-C ME meets the Level 2 security requirements of FIPS 140-2 for roles, services and authentication, the Level 3 security requirements of FIPS 140-2 for design assurance, and the Level 1 security requirements of FIPS 140-2 for all other aspects. It also describes how to securely operate Crypto-C ME in a FIPS 140-2-compliant manner.

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/.

To learn more about how Dell EMC products, services, and solutions can help solve your business and IT challenges, contact your local representative or authorized reseller, visit www.dellemc.com, or explore and compare products in the Dell EMC Store

Dell EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

The information in this publication is provided "as is." Dell EMC Corporation makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose.

TABLE OF 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 validated cryptographic module for Level 1 overall security, Level 2 roles, services, and authentication, and Level 3 design assurance.

1.1 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 4, 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 17 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 Dell EMC Corporation-proprietary and is releasable only under appropriate non-disclosure agreements. For access to these documents, please contact Dell EMC.

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 2 for roles, services, and authentication, Level 3 for design assurance, and Level 1 security requirements.

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	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	2
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 the following operating environments:

- Red Hat®:
    - Enterprise Linux 5.5, x86 (32-bit), built with LSB3.0.3 and gcc 3.4.6
    - Enterprise Linux 5.5, x86_64 (64-bit), built with LSB3.0.3 and gcc 3.4.6
    - Enterprise Linux 5.5, x86 (32-bit) with AES New Instructions (AES-NI), built with LSB3.0.3 and gcc 3.4.6
    - Enterprise Linux 6.0, x86 (32-bit), built with LSB3.0.3 and gcc 3.4.6
    - Enterprise Linux 6.0, x86_64 (64-bit), built with LSB3.0.3 and gcc 3.4.6

2.1.2 Affirmation of Compliance for other Operating Environments

Affirmation of compliance is defined in the Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program document, section G.5, (https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Module-Validation-Program/documents/fips140-2/FIPS1402IG.pdf). Compliance is maintained in all operational environments for which the binary executable remains unchanged. Specifically, Dell EMC affirms compliance for the following operational environments:

- Dell EMC Data Domain OS 5.7.2.0 and subsequent releases on x86_64 (64-bit)
- Dell EMC Data Domain OS 6.0.0.20 and subsequent releases on x86_64 (64-bit)

- Dell EMC Data Domain OS 7.0.0.9 and subsequent releases on x86_64 (64-bit)
- Dell EMC Data Domain OS releases under DD VE on x86_64 with the following hypervisors:
    - VMware
        - ESXi 5.5
        - ESXi 6
        - ESXi 6.5
        - ESXi 6.7
    - Microsoft Hyper V
        - Windows Server 2012
        - Windows Server 2016
    - KVM (KVM Hypervisor is supported only on Vanilla KVM. (Openstack, Ovirt, RHEV is not supported)
        - Ubuntu 14.04
        - Ubuntu 16.04
        - RHEL 7.3
        - RHEL 7.2
        - SUSE 12-SP2
        - CentOS 7

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.

2.1.3 Configuring Single User Mode

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

Data Domain OS or Linux

To configure single user mode for systems running a Data Domain OS or Linux 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`, setting the value of
   `disable` to `yes`.
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: `cryptocme.dll` (on systems running a Windows operating system), `libcryptocme.so` (on systems running a UNIX, Data Domain OS, Linux, or Solaris operating system), or `libcryptocme.sl` (on systems running an HP-UX operating system).

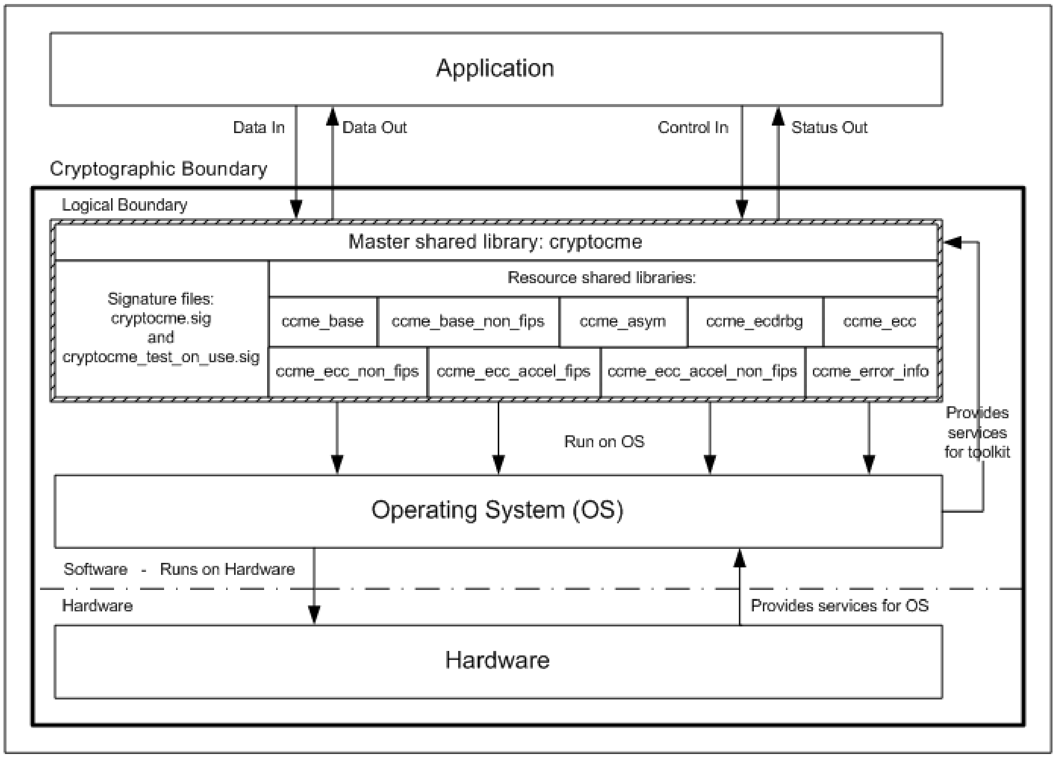
- Resource shared libraries:

- ccme_base.dll, ccme_base_non_fips.dll, ccme_asym.dll, ccme_ecdrbg.dll, ccme_ecc.dll, ccme_ecc_non_fips.dll, ccme_ecc_accel_fips.dll, ccme_ecc_accel_non_fips.dll, and ccme_error_info.dll on systems running a Windows operating system.
- 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 on systems running a UNIX, Linux, or Solaris operating system.
- libccme_base.sl, libccme_base_non_fips.sl, libccme_asym.sl, libccme_ecdrbg.sl, libccme_ecc.sl, libccme_ecc_non_fips.sl, libccme_ecc_accel_fips.sl, libccme_ecc_accel_non_fips.sl, and libccme_error_info.sl on systems running an HP-UX operating system.

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

The underlying logical interface to Crypto-C ME is the API, documented in the *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, Services, and Authentication

Crypto-C ME meets all FIPS 140-2 Level 2 requirements for roles, services, and authentication, implementing both a User role and Crypto Officer role. Role-based authentication is implemented for these roles. Only one role can be active at a time and Crypto-C ME does not allow concurrent operators.

2.3.1 Provider Configuration

The application is responsible for enabling Level 2 roles and authentication prior to the module being loaded.

The application must supply the `R_FIPS140_FEATURE_SL2_roles` feature when creating the FIPS 140 provider.

To load the cryptographic module with the `R_FIPS140_FEATURE_SL2_roles` feature:

1. Call `R_PROV_FIPS140_new() including R_FIPS140_FEATURE_SL2_roles` as one of the provider features.
2. Configure the location of the cryptographic module library files using `R_PROV_FIPS140_set_path()`.
3. `Call R_PROV_FIPS140_load()` to load the cryptographic module.

The cryptographic module uses a database of role identity information to validate authentication attempts by the operator. The roles database stores a salted message digest of a PIN for each role it authenticates. The roles database can be stored either in memory or in a file. The application must set up a roles database and add authentication data before it can perform Level 2 role authentication.

To create the roles database in a file:

1. Load the FIPS140 provider with the `R_FIPS140_FEATURE_SL2_roles` feature.
2. Set the location of the file by calling `R_PROV_FIPS140_set_roles_file()` and specify the path to the file.

   Note: For operating systems that use wide character sets, call `R_PROV_FIPS140_set_roles_file_w()` instead.
3. Create the file by calling `R_PROV_FIPS140_init_roles()`.

To create the roles database in memory:

1. Load the FIPS140 provider with the `R_FIPS140_FEATURE_SL2_roles` feature.
2. Initialize the data in memory by calling `R_PROV_FIPS140_init_roles()`.

To set the initial authentication data in the roles database, the call to `R_PROV_FIPS140_init_roles()` must supply an authentication callback function. The authentication callback function is called once for each type of role to allow the application to set the initial authentication data.

PIN Data

PIN data supplied by the application is hashed using the SHA-512 algorithm to generate 64 bytes of authentication data that is stored in the roles database.

To prevent successful random authentication attempts the application must set random PIN data of sufficient security strength. The PIN must contain random data with the equivalent of a minimum of 74 effectively random bits. The actual length of the PIN data depends upon the randomness of the source of PIN data used by the application.

The minimum length of the PIN specified in this Security Policy document is sufficient to prevent brute force searching of the PIN value with a probability greater than 1 in 100000 over a period of one minute when the hash calculations are performed by a computing resource with the performance equivalence of a cluster of up to one million Amazon EC2 GPU instances. Other considerations of the application environment might require other PIN lengths. Up to 64 bytes of PIN data can be passed to the module by a call to assume a role.

2.3.2 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`. The preinstalled authentication callback function will gather PIN data during the call. A message digest of the PIN, generated using SHA-512, is checked against the authentication data held by the roles database.

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 16.

2.3.3 Crypto User Role

An operator can assume the Crypto User role by calling `R_PROV_FIPS140_assume_role()` with `R_FIPS140_ROLE_USER`. The preinstalled authentication callback function will gather PIN data during the call. A message digest of the PIN, generated using SHA-512, is checked against the authentication data held by the roles database.

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 16.

2.3.4 Unloading and Reloading the Module

A roles database that is stored in memory is erased when the cryptographic module is unloaded. When the cryptographic module is reloaded, the roles database must be recreated before any roles are accessible. For the steps to create a roles database in memory, see "To create the roles database in memory".

A roles database that is stored in file remains on the file system when the module is unloaded. When the cryptographic module is reloaded, the application can reuse the existing roles database.

To reuse an existing roles database:

1. Load the FIPS140 provider with `the R_FIPS140_FEATURE_SL2_roles` feature.

2. Set the location of the file by calling `R_PROV_FIPS140_set_roles_file()` and specify the path to the file. This reads the roles database, if it exists.

   Note: For operating systems that use wide character sets, call `R_PROV_FIPS140_set_roles_file_w()`instead.

In all cases, when the module is reloaded the application cannot assume any role until it initializes access to the roles database. After access to the roles database is established an application must reauthenticate to each role it assumes.

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 no longer required.

2.4.1 Key Generation

Crypto-C ME uses an HMAC Deterministic Random Bit Generator (DRBG) in the generation of symmetric keys used in algorithms such as AES, Triple-DES 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

KEY OR CSP	STORAGE	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 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 *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
FIPS 186-4 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
- RSA key generation per FIPS 186-2
- RSA for signature generation per 186-2
- DSA for signature generation per 186-2
- ECDSA for signature generation per 186-2
- 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 13.

2.6 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.

2.6.1 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.

2.6.2 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).

2.6.3 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`.

2.6.4 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.

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 11. 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 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 16.
`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 16.

3.3 Modes of Operation

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

Table 6. 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 4on page 11.
`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.
`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 6 on page 14.

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 11 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 *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`. Authenticate and switch to a new role by calling `R_PROV_FIPS140_authenticate_role()` with one of the information identifiers listed in Table 6 on page 14.

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 *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 *Crypto-C Micro Edition API Reference Guide*.

4 Services

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

```
R_add()
BIO_append_filename()
BIO_cb_cmd_to_string()
BIO_cb_post(BIO_cb_pre()
BIO_CB_return()
BIO_clear_flags()
BIO_clear_retry_flags()
BIO_copy_next_retry()
BIO_ctrl()
BIO_debug_cb()
BIO_dump()
BIO_dump_format()
BIO_dup_chain()
BIO_dup_chain_ef()
BIO_eof()
BIO_f_buffer()
BIO_f_null()
BIO_find_type()
BIO_flags_to_string()
BIO_flush()
BIO_free()
BIO_free_all()
BIO_get_app_data()
BIO_get_buffer_num_lines()
BIO_get_cb()
BIO_get_cb_arg()
BIO_get_close()
BIO_get_flags()
BIO_get_fp()
BIO_get_info_cb()
BIO_get_mem_data()
BIO_get_retry_BIO()
BIO_get_retry_flags()
BIO_get_retry_reason()
BIO_get_state_cb()
BIO_get_state_cb_arg()
BIO_gets()
BIO_method_name()
BIO_method_type()
BIO_new()
BIO_new_ef()
BIO_new_file()
BIO_new_file_ef()
BIO_new_file_w()
BIO_new_file_w_ef()
BIO_new_fp()
BIO_new_fp_ef()
BIO_new_init()
BIO_new_init_ef()
BIO_new_mem()
BIO_new_mem_ef()
BIO_open_file()
BIO_open_file_w()
BIO_pending()
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_set_app_data()
BIO_set_bio_cb()
BIO_set_buffer_read_data()
BIO_set_buffer_size()
BIO_set_cb()
BIO_set_cb_arg()
BIO_set_cb_recursive()
BIO_set_close()
BIO_set_flags()
BIO_set_fp()
BIO_set_info_cb()
BIO_set_mem_eof_return()
BIO_set_read_buffer_size()
BIO_set_retry_read()
BIO_set_retry_small_buffer()
BIO_set_retry_special()
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 7. 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.
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.

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.
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.