18.01.17

RSA BSAFE® Crypto-C Micro Edition

Version 4.1, 4.1.0.1, and 4.1.2
Security Policy Level 1

This is a non-proprietary Security Policy for RSA BSAFE Crypto-C Micro Edition (Crypto-C ME) 4.1, 4.1.0.1, and 4.1.2. 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 the cryptographic module specification and 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, see the FIPS 140-2 page on the NIST Web site at `http://csrc.nist.gov/groups/STM/cmvp/standards.html`.

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

 Copyright © 2017 EMC Corporation. All rights reserved. Published in the USA.

1 Introduction

The Crypto-C ME software development toolkit is designed to enable developers to incorporate cryptographic technologies into applications. Crypto-C ME security software helps to protect sensitive data as it is stored, using strong encryption techniques to 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 the following:

- Information on the full line of RSA products and services is available at `https://www.rsa.com/en-us`.
- RSA BSAFE product overviews, technical information, and answers to sales-related questions are available at `https://community.rsa.com/community/products/bsafe`.

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 26 specifically addresses the required configuration for the FIPS 140-2 mode of operation.
- "Services" on page 31 lists the functions of Crypto-C ME.
- "Acronyms and Definitions" on page 37 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

Crypto-C ME is designed with the ability to optimize code for different processors, and specific speed or size requirements. Assembly-level optimizations on key processors mean 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 containing 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 and Services	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. This section lists the operating environments Crypto-C ME 4.1 and 4.1.2 are tested on.

Crypto-C ME 4.1

Crypto-C ME 4.1 is tested on the following operating environments:

- Apple®:
  - Mac® OS X 10.x on x86_64 (64-bit), built with gcc 4.2.1
  - iOS 6.x on ARMv7 (32-bit), built with Xcode 5 and clang 500.2.76
- Canonical® Ubuntu® 12.04 Long Term Support (LTS) on ARMv7, built with gcc 4.6 (hard float)
- FreeBSD® 8.3 on x86_64 (64-bit), built with gcc 4.2
- Google® Android®:
  - 2.3 (Gingerbread) on ARMv7 (32-bit), built with Android NDK rev 8d and gcc 4.6.7
  - 4.0 (Ice Cream Sandwich) on x86 (32-bit), built with Android NDK rev 8d and gcc 4.6.7
  - 4.1 (Jelly Bean) on ARMv7 (32-bit), built with Android NDK rev 8d and gcc 4.6.7

- HP:
  - HP-UX 11.31 on:
    - PA-RISC 2.0 (32-bit), built with HP ANSI-C 11
    - PA-RISC 2.0W (64-bit), built with HP ANSI-C 11
    - Itanium (32-bit), built with cc B3910B A.06.12
    - Itanium (64-bit), built with cc B3910B A.06.12
- IBM® AIX®:
  - v6.1 on:
    - PowerPC (32-bit), built with XLC v9.0
    - PowerPC (64-bit), built with XLC v9.0
  - v7.1 on:
    - PowerPC (32-bit), built with XLC v11.1
    - PowerPC (64-bit), built with XLC v11.1
- Micro Focus®:
  - SUSE® Linux Enterprise Server 11.0 on:
    - x86 (32-bit), built with LSB4.0 and gcc 4.4
    - x86_64 (64-bit), built with LSB4.0 and gcc 4.4
    - PowerPC (32-bit), built with gcc 3.4
    - PowerPC (64-bit), built with gcc 3.4
- Microsoft® Windows:
  - 7 Enterprise SP1 on:
    - x86 (32-bit), built with Visual Studio 2005, no C runtime library (no CRT)
    - x86 (32-bit), built with Visual Studio 2010
    - x86-64 (64-bit), built with Visual Studio 2005
    - x86-64 (64-bit), built with Visual Studio 2010, no CRT
  - 8.1 Enterprise, x86-64 (64-bit), built with Visual Studio 2010, no CRT
  - Server 2003 Enterprise R2 on:
    - x86 (32-bit), built with Visual Studio 2005, no CRT
    - x86 (32-bit), built with Visual Studio 2010
    - x86_64 (64-bit), built with Visual Studio 2005
    - x86_64 (64-bit), built with Visual Studio 2010, no CRT
    - Itanium® (64-bit), built with Visual Studio 2005
    - Itanium (64-bit), built with Visual Studio 2010, no CRT

- Server 2008 Enterprise R2 on:
  - x86 (32-bit), built with Visual Studio 2005, no CRT
  - x86 (32-bit), built with Visual Studio 2010
  - x86_64 (64-bit), built with Visual Studio 2005
  - x86_64 (64-bit), built with Visual Studio 2010, no CRT
  - Itanium (64-bit), built with Visual Studio 2005
  - Itanium (64-bit), built with Visual Studio 2010, no CRT
- Server 2012 Standard, x86_64 (64-bit), built with Visual Studio 2010
- Server 2012 Standard R2, x86_64 (64-bit), built with Visual Studio 2010, no CRT

- Oracle® Solaris®:
  - 10 on:
    - SPARC® v8 (32-bit), built with Sun Studio 10, Sun C 5.8
    - x86 (32-bit), built with Sun Studio 10, Sun C 5.8
    - x86_64 (64-bit), built with Sun Studio 10, Sun C 5.8
  - 11 on:
    - SPARC v8+ (32-bit), built with Solaris Studio 12.3, Sun C 5.12
    - SPARC v9-T2 (64-bit), built with Solaris Studio 12.3, Sun C 5.12
    - SPARC v9-T4 (64-bit), built with Solaris Studio 12.3, Sun C 5.12
- Red Hat®:
  - Enterprise Linux® 5.x on:
    - x86 (32-bit), built with Linux Standard Base (LSB) 3.0 and gcc 3.4
    - x86_64 (64-bit), built with LSB3.0 and gcc 3.4
    - Itanium (64-bit), built with LSB3.0 and gcc 3.4
    - PowerPC® (32-bit), built with gcc 3.4
    - PowerPC (64-bit), built with gcc 3.4
    - IBM® S390 (31-bit), built with gcc 4.3
    - IBM S390x (64-bit), built with gcc 4.3
  - Enterprise Linux 6.x on:
    - x86 (32-bit), built with LSB4.0 and gcc 4.4
    - x86_64 (64-bit), built with LSB4.0 and gcc 4.4
  - Fedora™ 17 on ARMv7, built with gcc 4.6 (soft float)
- WindRiver® VxWorks:
  - 6.4 on PowerPC 604 (32-bit), built with gcc version 3.4.4
  - 6.7 on PowerPC 604 (32-bit), built with gcc version 4.1.2

Crypto-C ME 4.1.0.1

- Linaro Linux 3.10.68 on ARMv7 (32-bit), built with gcc version 4.8.3.

Crypto-C ME 4.1.2

Crypto-C ME 4.1.2 is tested on the following operating environments:

- Apple Mac OS X 10.10 on x86_64 (64-bit), built with Xcode 7
- Canonical Ubuntu 12.04 Long Term Support (LTS) on ARMv7, built with gcc 4.6 (hard float)
- FreeBSD 10 on x86_64 (64-bit), built with gcc 4.2
- Google Android:
  - 4.1 on x86 (32-bit), built with Android NDK r10e and gcc 4.9
  - 4.4 on ARMv7 (32-bit), built with Android NDK r10e and gcc 4.9
  - 5.1 on:
    - ARMv7 (32-bit), built with Android NDK r10e and gcc 4.9
    - ARM64 (64-bit), built with Android NDK r10e and gcc 4.9
- HP HP-UX 11.31 on:
  - PA-RISC 2.0(32-bit), built with HP ANSI-C 11.11.12
  - PA-RISC 2.0W (64-bit), built with HP ANSI-C 11.11.12
  - Itanium (32-bit) and Itanium (64-bit), built with cc B3910B A.06.12
- IBM AIX:
  - v6.1 on PowerPC (32-bit) and PowerPC (64-bit), built with XLC v9.0
  - v7.1 on PowerPC (32-bit) and PowerPC (64-bit), built with XLC v11.1
- Micro Focus SUSE Linux Enterprise Server:
  - 11 SP4 on:
    - x86 (32-bit) and x86_64 (64-bit), built with Linux Standard Base (LSB) 4.0 and gcc 4.4
    - PowerPC (32-bit) and Power PC (64-bit), built with gcc 3.4
  - 12 on x86 (32-bit) and x86_64 (64-bit) built with LSB 4.0 and gcc 4.4
- Microsoft Windows:
  - 7 Enterprise SP1 on:
    - x86 (32-bit), built with Visual Studio 2005 (/MT[1])
    - x86-64 (64-bit), built with Visual Studio 2005 (/MD)
    - x86-64 (64-bit), built with Visual Studio 2010 (/MT)
  - 8 Enterprise on x86 (32-bit), built with Visual Studio 2013 (/MT)
  - 8.1 Enterprise on x86-64 (64-bit), built with Visual Studio 2010 (/MT)

[1] Multi-threaded dynamic linked runtime library (MD) and multi-threaded static linked runtime library (MT).

  - 10 Enterprise on:
    - x86 (32-bit), built with Visual Studio 2013 (/MD)
    - x86-64 (64-bit), built with Visual Studio 2013 (/MD)
  - Server 2008 Enterprise SP2 on:
    - x86 (32-bit), built with Visual Studio 2005 (/MT)
    - x86 (32-bit), built with Visual Studio 2010 (/MD)
    - Itanium (64-bit), built with Visual Studio 2005 (/MD)
    - Itanium (64-bit), built with Visual Studio 2010 (/MT)
  - Server 2008 Enterprise R2 SP1 on:
    - x86-64 (64-bit), built with Visual Studio 2005 (/MD)
    - x86-64 (64-bit), built with Visual Studio 2010 (/MT)
  - Server 2012 Standard R2 on:
    - x86_64 (64-bit), built with Visual Studio 2010 (/MT)
    - x86_64 (64-bit), built with Visual Studio 2013 (/MD)
- Oracle Solaris:
  - 10 on SPARC v8 (32-bit), built with Sun C 5.8
  - 10 Update 11 on:
    - SPARC v8 (32-bit), built with Sun C 5.8
    - x86 (32-bit), built with Sun C 5.13
    - x86_64 (64-bit), built with Sun C 5.13
  - 11.2 on:
    - SPARC v8+ (32-bit), built with Sun C 5.13
    - SPARC v9-T2 (64-bit), built with Sun C 5.13
    - SPARC v9-T4 (64-bit), built with Sun C 5.13
- Red Hat:
  - Enterprise Linux 5.x on:
    - IBM S390 (31-bit), built with gcc 4.3
    - IBM S390x (64-bit), built with gcc 4.3
  - Enterprise Linux 5.11 on:
    - x86 (32-bit) and x86_64 (64-bit), built with LSB3.0 and gcc 3.4
    - PowerPC (32-bit) and PowerPC (64-bit), built with gcc 3.4
  - Enterprise Linux 6.7 on x86 (32-bit) and x86_64 (64-bit), built with LSB 4.0 and gcc 4.4
  - Enterprise Linux 7.1 on x86 (32-bit) and x86_64 (64-bit), built with LSB4.0 and gcc 4.4

- Fedora™ 20 on ARMv7 (32-bit), built with gcc 4.6 (hard float)
- Fedora 22 on ARM64 (64-bit), built with gcc 4.9
- CentOS 7.2 on x86_64 (64-bit), built with LSB 4.0 and gcc 4.4

- Wind River VxWorks:
  - 6.4 on PowerPC 604 (32-bit)
  - 6.7 on PowerPC 604 (32-bit)
  - 6.8 on ARMv4 (32-bit).

2.1.2 Affirmation of Compliance for other Operating Environments

Affirmation of compliance is defined in Section G.5, “Maintaining Validation Compliance of Software or Firmware Cryptographic Modules,” in *Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program*. Compliance is maintained in all operational environments for which the binary executable remains unchanged.

The Cryptographic Module Validation Program (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.

Crypto-C ME 4.1

For Crypto-C ME 4.1, RSA affirms compliance for the following operating environments:

- Apple:
  - Mac OS X 10.x on x86 (32-bit), built with gcc 4.0.1
  - iOS 6.x on ARMv7s (32-bit), built with Xcode 5 and clang 500.2.76
- Canonical:
  - Ubuntu 11.04 on ARMv7, built with gcc 4.6 (soft float)
  - Ubuntu 12.04 LTS on:
    - x86 (32-bit), built with LSB 3.0 and gcc 3.4
    - x86_64 (64-bit), built with LSB3.0 and gcc 3.4
    - x86_64 (64-bit), built with LSB4.0 and gcc 4.4
- HP HP-UX 11.23 on:
  - PA-RISC 2.0 (32-bit), built with HP ANSI-C 11
  - PA-RISC 2.0W (64-bit), built with HP ANSI-C 11
- IBM AIX v5.3 on:
  - PowerPC (32-bit), built with XLC v8.0
  - PowerPC (64-bit), built with XLC v8.0

- Micro Focus SUSE Linux Enterprise Server:
  - 10 on:
    - x86 (32-bit), built with LSB 3.0 and gcc 3.4
    - x86_64 (64-bit), built with LSB3.0 and gcc 3.4
    - PowerPC (32-bit), built with gcc 3.4
    - PowerPC (64-bit), built with gcc 3.4
  - 11 on Itanium (64-bit), built with LSB3.0 and gcc 3.4
- Microsoft Windows:
  - XP Professional, SP3 on x86 (32-bit), built with Visual Studio 2005 and Visual Studio 2010, either CRT or no CRT
  - XP Professional, SP2 on x86_64 (64-bit), built with Visual Studio 2005 and Visual Studio 2010, either CRT or no CRT
  - Vista Enterprise on x86 (32-bit), built with Visual Studio 2005 and Visual Studio 2010, either CRT or no CRT
  - Vista Enterprise on x86_64 (64-bit), built with Visual Studio 2005 and Visual Studio 2010, either CRT or no CRT
  - 7 Enterprise, SP1 on x86 (32-bit), built with Visual Studio 2005 and Visual Studio 2010, either CRT or no CRT
  - 7 Enterprise, SP1 on x86_64 (64-bit), built with Visual Studio 2005 and Visual Studio 2010, either CRT or no CRT
  - 8.1 Enterprise on x86_64 (64-bit), built with Visual Studio 2010, either CRT or no CRT
  - Server 2003 Enterprise, R2 on Itanium, built with Visual Studio 2005 and Visual Studio 2010, either CRT or no CRT
  - Server 2008 Enterprise, R2 on Itanium, built with Visual Studio 2005 and Visual Studio 2010, either CRT or no CRT.
  - Server 2012 Standard, x86_64 (64-bit), built with Visual Studio 2010, no CRT
  - Server 2012 Standard R2, x86_64 (64-bit), built with Visual Studio 2010
- Oracle Solaris:
  - 10 on:
    - SPARC v8+ (32-bit), built with Solaris Studio 12.3, Sun C 5.12
    - SPARC v9-T2 (64-bit), built with Solaris Studio 12.3, Sun C 5.12
    - SPARC v9-T4 (64-bit), built with Solaris Studio 12.3, Sun C 5.12
    - SPARC v9-T5 (64-bit), built with Solaris Studio 12.3, Sun C 5.12
  - 11 on SPARCv9-T5 (64-bit), built with Solaris Studio 12.3, Sun C 5.12.
- Red Hat Enterprise Linux 5.x, Security Enhanced (SE) configuration on:
  - x86 (32-bit), built with LSB 3.0 and gcc 3.4
  - x86_64 (64-bit), built with LSB3.0 and gcc 3.4
- WindRiver VxWorks 6.8 on ARMv4 (32-bit), built with gcc version 4.1.2.

Crypto-C ME 4.1.2

For Crypto-C ME 4.1.2, RSA affirms compliance for the following operating environments:

- Apple Mac OS X:
  - 10.10 on x86 (32-bit), built with Xcode 7
  - 10.11 on x86 (32-bit) and x86_64 (64-bit), built with Xcode 7
- Canonical Ubuntu 12.04 LTS on:
  - x86 (32-bit) and x86_64 (64-bit), built with LSB 3.0 and gcc 3.4
  - x86 (32-bit) and x86_64 (64-bit), built with LSB 4.0 and gcc 4.4.
- Google Android 6.0 on ARMv7 (32-bit), built with Android NDK r10e and gcc 4.9
- Micro Focus SUSE Linux Enterprise Server:
  - 10 on:
    - x86 (32-bit) and x86_64 (64-bit), built with LSB 3.0 and gcc 3.4
    - PowerPC (32-bit) and PowerPC (64-bit), built with gcc 3.4
  - 11 SP2 on x86_64 (64-bit), built with LSB 4.0 and gcc 4.4
  - 11 SP4 on Itanium (64-bit), built with LSB 3.0 and gcc 3.4
- Microsoft:
  - Windows Vista Enterprise on x86 (32-bit), built with Visual Studio 2005 or Visual Studio 2010, either /MD[2] or /MT
  - Windows Vista Enterprise on x86_64 (64-bit), built with Visual Studio 2005 , either /MD or /MT
  - Windows Vista Enterprise R2 on x86_64 (64-bit), built with Visual Studio 2010, either /MD or /MT
  - Windows 7 Enterprise, SP1 on x86 (32-bit), built with Visual Studio 2005, /MD
  - Windows 7 Enterprise, SP1 on x86 (32-bit), built with Visual Studio 2010, either /MD or /MT
  - Windows 7 Enterprise, SP1 on x86_64 (64-bit), built with Visual Studio 2005, /MT
  - Windows 7 Enterprise, SP1 on x86_64 (64-bit), built with Visual Studio 2010, /MD
  - Windows 8 Enterprise on x86 (32-bit), built with Visual Studio 2013, /MD
  - Windows 8 Enterprise on x86_64 (64-bit), built with Visual Studio 2013, either /MD or /MT
  - Windows 8.1 Enterprise on x86_64 (64-bit), built with Visual Studio 2010, /MD
  - Windows 8.1 Enterprise on x86_64 (64-bit), built with Visual Studio 2013, /MD or /MT
  - Windows 10 Enterprise on x86 (32-bit), built with Visual Studio 2013, /MT

[2] Multi-threaded dynamic linked runtime library (MD) and multi-threaded static linked runtime library (MT).

- Windows 10 Enterprise on x86_64 (64-bit), built with Visual Studio 2013, /MD
- Windows Server 2008 Enterprise SP2 on x86 (32-bit), built with Visual Studio 2005, /MD
- Windows Server 2008 Enterprise SP2 on x86 (32-bit), built with Visual Studio 2010, /MT
- Windows Server 2008 Enterprise SP2 on Itanium (64-bit), built with Visual Studio 2005, /MT
- Windows Server 2008 Enterprise SP2 on Itanium (64-bit), built with Visual Studio 2010, /MD
- Windows Server 2008 Enterprise R2 SP1 on x86_64 (64-bit), built with Visual Studio 2005, /MT.
- Windows Server 2008 Enterprise R2 SP1 on x86_64 (64-bit), built with Visual Studio 2010, /MD.
- Windows Server 2012 Standard on x86_64 (64-bit), built with Visual Studio 2010 or Visual Studio 2013, /MD or /MT
- Windows Server 2012 Standard R2 on x86_64 (64-bit), built with Visual Studio 2010, /MD
- Windows Server 2012 Standard R2 on x86_64 (64-bit), built with Visual Studio 2013, /MT

- Oracle Solaris:
  - 10 Update 11 on:
    - SPARC v8+ (32-bit), built with Sun C 5.13
    - SPARC v9-T2 (64-bit), built with Sun C 5.13
    - SPARC v9-T4 (64-bit), built with Sun C 5.13
    - SPARC v9-T5 (64-bit), built with Sun C 5.13
  - 11 on SPARCv8 (32-bit), built with Sun C 5.8
- Red Hat:
  - Enterprise Linux 5.x Security Enhanced (SE) configuration, x86 (32-bit) and x86_64 (64-bit), built with LSB 3.0 and gcc 3.4
  - Enterprise Linux 6.x on x86 (32-bit) and x86_64 (64-bit), built with LSB 3.0 and gcc 3.4
  - Enterprise Linux 7.x on PowerPC (32-bit) and PowerPC (64-bit), built with and gcc 3.4
  - CentOS 6.6 on x86_64 (64-bit), built with LSB 4.0 and gcc 4.4

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.

Apple Mac OS X

To configure single user mode for systems running an Apple Mac OS X operating system:

1. Start a terminal session.
2. Edit `/etc/passwd` and `/etc/master.passwd` to remove all the users except `root` and the pseudo-users (daemon users). Make sure the password fields in `/etc/master.passwd` for the pseudo-users are either a star (`*`) or double exclamation mark (`!!`). This prevents login as the pseudo-users.
3. Disable the following services: `exec`, `ftp`, `login`, `shell`, `telnet`, and `tftp`. To do this from the command line:

   ```
   sudo launchctl unload -w /System/Library/LaunchDaemons/
      <service_name>.plist
   ```
4. Delete user accounts.
   a. Run **System Preferences**.
   b. Select **Accounts**.
   c. Click on the lock to make changes and authenticate yourself.
   d. Delete all user accounts except your account.
5. Disable services.
   a. Run **Directory Utility**.
   b. Select **Show Advanced Settings**.
   c. Select the **Service** tab.
   d. Click on the lock to make changes and authenticate yourself.
   e. Disable all services other than **Local**.
6. Reboot the system for the changes to take effect.

Apple iOS

The Apple iOS operating system is a single user operating system so no steps are required to configure single user mode.

FreeBSD

To configure single user mode for systems running a FreeBSD 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 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.

Google Android

The Google Android operating systems are single user operating systems so no steps are required to configure single user mode.

HP-UX

To configure single user mode for systems running an HP-UX operating system:

1. Log in as the `root` user.
2. Edit `/etc/passwd` and remove all the users except root and the pseudo-users. Make sure the password fields for the pseudo-users are a star (`*`). This prevents login as the pseudo-users.
3. Edit `/etc/nsswitch.conf` so files is the only option for `passwd` and `group`. This disables the Network Information Service (NIS) and other name services for users and groups.
4. Edit `/etc/inetd.conf` to remove or comment out the lines for remote login, remote command execution, and file transfer daemons such as `telnetd`, `rlogind`, `remshd`, `rexecd`, `ftpd`, and `tftpd`.
5. Reboot the system for the changes to take effect.

IBM AIX

To configure single user mode for systems running an IBM AIX operating system:

1. Log in as the `root` user.
2. Edit `/etc/passwd` and remove all the users except `root` and the pseudo-users. Make sure the password fields for the pseudo-users are a star (`*`). This prevents login as the pseudo-users.
3. Remove all lines beginning with a plus sign (`+`) or minus sign (`-`) from `/etc/passwd` and `/etc/group`. This disables the Network Information Service (NIS) and other name services for users and groups.
4. Edit `/etc/inetd.conf` to remove or comment out the lines for remote login, remote command execution, and file transfer daemons such as `telnetd`, `rlogind`, `remshd`, `rexecd`, `ftpd`, and `tftpd`.
5. Reboot the system for the changes to take effect.

Microsoft Windows

To configure single user mode for systems running a Microsoft Windows XP Professional, Windows Vista Enterprise, Windows 7 Enterprise, Windows 8 Enterprise, Windows 2003 Server Enterprise, Windows 2008 Server Enterprise, or Windows 2012 Server Standard operating system, guest accounts, server services, terminal services, remote registry services, remote desktop services, and remote assistance must be disabled. For detailed instructions on how to perform these tasks, see the Microsoft support site.

Oracle Solaris

To configure single user mode for systems running an 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 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.

Red Hat Enterprise, Fedora, CentOS, Micro Focus SUSE, Canonical Ubuntu, or Linaro Linux

To configure single user mode for systems running a 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 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.

Wind River VxWorks

The Wind River VxWorks operating systems are single user operating systems so no steps are required to configure single user mode.

2.2 Crypto-C ME Interfaces

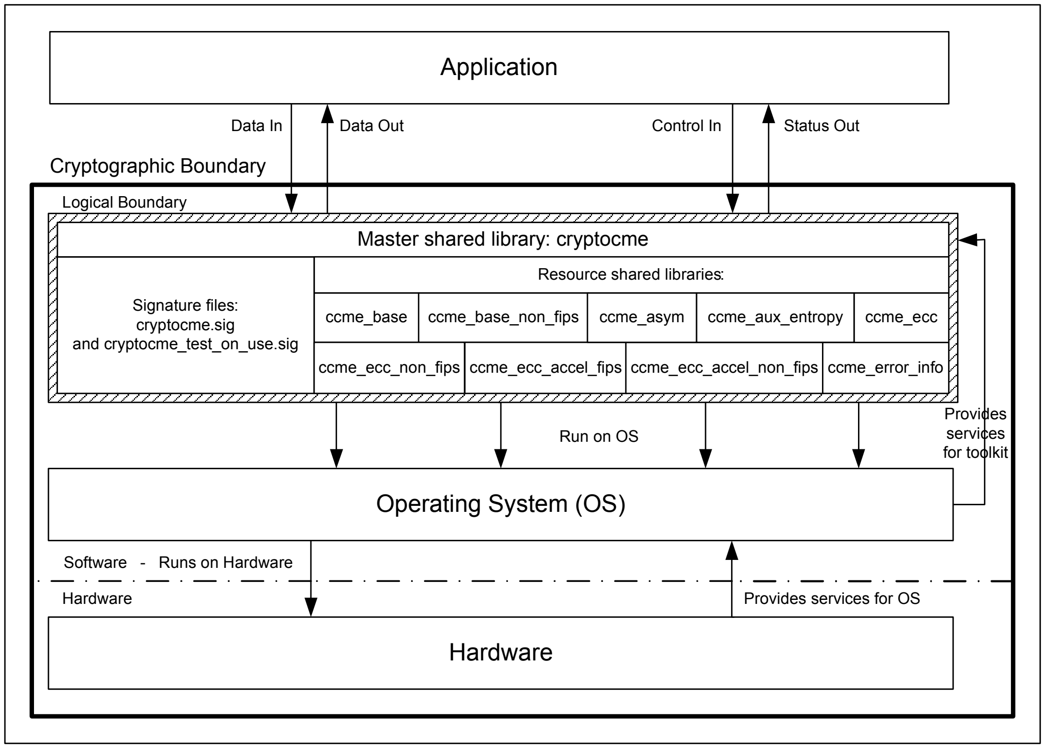
Crypto-C ME is validated as a multi-chip standalone cryptographic module. The physical cryptographic boundary of the module is the case of the general-purpose computer or mobile device, which encloses the hardware running the module. The physical interfaces for Crypto-C ME consist of the keyboard, mouse, 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 comprising the module:

- Master shared library:
  - `cryptocme.dll` on systems running a Windows operating system
  - `libcryptocme.so` on systems running a Solaris, Linux, AIX, FreeBSD, or Android, or VxWorks operating system
  - `libcryptocme.sl` on systems running an HP-UX operating system
  - `libcryptocme.dylib` on systems running a Mac OS X or iOS operating system.
- Resource shared libraries:
  - `ccme_base.dll`, `ccme_base_non_fips.dll`, `ccme_asym.dll`, `ccme_aux_entropy.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_aux_entropy.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 Solaris, Linux, AIX, FreeBSD, or Android operating system.
  - `libccme_base.sl`, `libccme_base_non_fips.sl`, `libccme_asym.sl`, `libccme_aux_entropy.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.
  - `libccme_base.dylib`, `libccme_base_non_fips.dylib`, `libccme_asym.dylib`, `libccme_aux_entropy.dylib`, `libccme_ecc.dylib`, `libccme_ecc_non_fips.dylib`, `libccme_ecc_accel_fips.dylib`, `libccme_ecc_accel_non_fips.dylib`, and `libccme_error_info.dylib` on systems running a Mac OS X or iOS 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 *RSA BSAFE Crypto-C Micro Edition Developers Guide*. Crypto-C ME provides for Control Input through the API calls. Data Input and Output are provided by the variables passed with the API calls, and Status Output is provided through the returns and error codes documented for each call. This is illustrated in the following diagram.

Figure 1 Crypto-C ME Logical Interfaces



Note: Shared libraries for systems running a Mac OS X or iOS operating system might include Apple code signatures applied by customers. If such a signature is present, the signature is not included in the logical boundary and is explicitly excluded from the software signature check.

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 31.

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 31.

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 the generation of DSA, RSA, Diffie-Hellman (DH) and Elliptic Curve Cryptography (ECC) public and private keys. Also, Crypto-C ME uses the CTR Deterministic Random Bit Generator (CTR DRBG) as the default pseudo-random number generator (PRNG) for asymmetric and symmetric keys used in algorithms such as AES, Triple DES, RSA, DSA, 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	Generation/Input	Storage
Hardcoded DSA public key (2048-bit)	Generated when the module is created	Persistent storage embedded in the module binary (encrypted)
Hardcoded AES key (128-bit)	Generated when the module is created	Persistent storage embedded in the module binary (plaintext)
AES keys (128, 192, and 256-bit key sizes)	Entered in plaintext through the API	Volatile memory only (plaintext)
Triple-DES keys (192-bit key size)	Entered in plaintext through the API	Volatile memory only (plaintext)
HMAC with SHA-1 and SHA-2 keys (greater than 112-bit key size)	Entered in plaintext through the API	Volatile memory only (plaintext)
Diffie-Hellman public/private keys (2048 to 4096-bit key sizes)	Entered in plaintext through the API	Volatile memory only (plaintext)
ECC public/private keys (224 to 571-bit key sizes, less than 224 bits for legacy signature verification only)	Entered in plaintext through the API	Volatile memory only (plaintext)
RSA public/private keys (2048 to 4096-bit key sizes, less than 2048 bits for legacy signature verification only)	Entered in plaintext through the API	Volatile memory only (plaintext)

Table 2 Key Storage (continued)

Key or CSP	Generation/Input	Storage
DSA public/private keys (2048 to 4096-bit key sizes, less than 2048 bits for legacy signature verification only)	Entered in plaintext through the API	Volatile memory only (plaintext)
CTR DRBG entropy (128 bits)	Generated internally	Volatile memory only (plaintext)
CTR DRBG V value (128 bits)	Generated internally	Volatile memory only (plaintext)
CTR DRBG key (256 bits)	Generated internally	Volatile memory only (plaintext)
CTR DRBG init_seed (384 bits)	Generated internally	Volatile memory only (plaintext)
HMAC DRBG entropy (112 to 256 bits)	Generated internally	Volatile memory only (plaintext)
HMAC DRBG V value (160 to 512 bits)	Generated internally	Volatile memory only (plaintext)
HMAC DRBG key (160 to 512 bits)	Generated internally	Volatile memory only (plaintext)
HMAC DRBG init_seed (440 to 888 bits)	Generated internally	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 Type	Key or CSP	Type of Access
Encryption and decryption	Symmetric keys (AES, Triple-DES)	Read/Execute
Digital signature and verification	Asymmetric keys (DSA, Elliptic Curve DSA (ECDSA), and RSA)	Read/Execute
Message digest	None	N/A
MAC	HMAC keys	Read/Execute
Random number generation	CTR DRBG entropy, V, key, and init_seed HMAC DRBG entropy, V, key, and init_seed	Read/Write/Execute

Table 3 Key and CSP Access (continued)

Service Type	Key or CSP	Type of Access
Key generation	Symmetric keys (AES, Triple-DES) Asymmetric keys (DSA, ECDSA, RSA, Diffie-Hellman (DH), and 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. All volatile keys and CSPs listed in Table 2 are zeroized by unloading the module from memory.

2.5 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 and allowed algorithms supported by Crypto-C ME, with their appropriate standards and validation certificate numbers.

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

			Validation Certificate		
Algorithm Type	**Algorithm**	**Standard**	**4.1**	**4.1.0.1**	**4.1.2**
Symmetric Key	AES in CBC, CFB 128-bit, ECB, OFB 128-bit, and CTR modes (with 128, 192, and 256-bit key sizes)	NIST SP800-38A	2859	3767	3596
	AES in CCM modes (with 128, 192, and 256-bit key sizes)	NIST SP800-38C			
	AES in GCM mode with automatic Initialization Vector (IV) generation (with 128, 192, and 256-bit key sizes).	NIST SP800-38D			
	AES in XTS mode[1] (with 128 and 256-bit key sizes)	NIST SP800-38E			
	Triple-DES[2] in ECB, CBC, CFB 64-bit, and OFB 64-bit modes.	NIST SP800-67 and SP800-38A	1706	2095	2003
Asymmetric Key	DSA (2048 to 4096-bit key sizes)	NIST FIPS 186-4	858	1047	999
	ECDSA (224 to 571-bit key sizes; curves tested: P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, and B-571) ECDSA2 Component Test	NIST FIPS 186-4	507 299	810 740	733 621
	RSA (2048 to 4096-bit key size) - Key generation and signature - Signature, verification only: RSASP1 Component Test RSADP Component Test	 NIST FIPS 186-4 NIST FIPS 186-2	1499 298 300	1938 716 717	1850 622 620
Key Agreement	DH (2048 to 4096-bit key size) and ECDH (224 to 571-bit key size) - Parameter generation: - Key generation:	 IEEE P1363 draft 10 Section A.16.1 RSA PKCS #3	Non-approved (Allowed in FIPS 140-2 mode).		
	KASECC_(ECC CDH) Primitive Component Test (curves tested: P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, and B-571)	SP800-56A Section 5.7.1.2	296	715	618
Key Derivation Functions (KDFs)	X9.63 KDF - Component Test	ANSI X9.63	297	714	619
	TLS Pseudo-random Function (TLS PRF) - Component Test	SP 800-135 rev1	297	714	619
	Password-based Key Derivation Function 2 (PBKDF2)[3]	SP 800-132	Vendor affirmed[4]		
Key Wrap	AES key wrap (with 128, 192, and 256-bit key sizes)	SP 800-38F	Non-approved[5]		3596
	AES padded key wrap (with 128, 192, and 256-bit key sizes)	SP 800-38F	Non-approved		3596
	RSA encrypt and decrypt (2048 to 4096-bit key size)[6]	PKCS #1	Non-approved (Allowed in FIPS 140-2 mode for key transport).		

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

			Validation Certificate		
Algorithm Type	**Algorithm**	**Standard**	**4.1**	**4.1.0.1**	**4.1.2**
Random Number	CTR DRBG	SP 800-90A rev1	507	1037	931
	HMAC DRBG (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512)	SP 800-90A rev1	507	1037	931
Message Digest	SHA-1, and SHA-224, 256, 384, 512, 512/224, and 512/256	FIPS180-4	2402	3137	2958
Message Authentication Code	HMAC-SHA1, SHA224, SHA256, SHA384, SHA512, SHA512/224, and SHA512/256	FIPS198-1	1799	2467	2293

[1]AES in XTS mode is only approved for hardware storage applications.

[2]Two-key Triple-DES encryption is not allowed. Two-key Triple-DES decryption is allowed for legacy-use only.

[3]As defined in NIST Special Publication 800-132, PBKDF2 can be used in FIPS 140-2 mode when used with FIPS 140-2-approved symmetric key and message digest algorithms. For more information, see "Crypto Officer and Crypto User Guidance" on page 26.

[4]Not yet tested by the CAVP, but is approved for use in FIPS 140-2 mode. RSA affirms correct implementation of the algorithm.

[5]For the 4.1 and 4.1.0.1 validations, CAVP testing was not available for SP800-38F AES key wrap and padded key wrap. Therefore, the AES key wrap implementations do not have CAVP certification and are considered to be non-approved but allowed in FIPS 140-2 mode for versions 4.1 and 4.1.0.1 of the module.

[6]For key wrapping using RSA, the key establishment methodology provides between 112 and 150 bits of encryption strength. Less than 112 bits of security (key sizes less than 2048 bits) is non-compliant.

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

- AES in CFB 64-bit and CTS modes
- AES in BPS mode for format-preserving encryption (FPE)
- DES
- DESX
- DES40
- Camellia
- GOST
- SEED
- RC2
- RC4
- RC5
- RSA with key sizes less than 2048 bits
- DSA with key sizes less than 2048 bits
- ECDSA with key sizes less than 224 bits
- DH with key sizes less than 2048 bits
- ECDH with key sizes less than 224 bits
- MD2
- MD4

- MD5
- HMAC MD5
- ECAES
- ECIES
- 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 26.

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 the 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.

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

2.6.1 Power-up Self-test

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

- AES in CCM, GCM, GMAC, and XTS mode Known Answer Tests (KATs) (encrypt/decrypt)
- Triple DES KATs (encrypt/decrypt)
- SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 KATs
- HMAC SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 SHA-512/224, and SHA-512/256 KATs
- TLS 1.0/1.1 ANSI X9.63 KDF KATs
- RSA sign/verify KATs
- RSA sign/verify test
- DSA sign/verify test
- ECDSA sign/verify test
- DH and ECDH conditional tests

- PRNG (CTR DRBG and HMAC DRBG) KATs
- Software integrity test using DSA signature verification.

Power-up self-tests are executed automatically when Crypto-C ME loads 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 a DSA, RSA, or EC public/private key pair.
- A Continuous Random Number Generation (CRNG) test each time the toolkit produces random data, as per the FIPS 140-2 standard. The CRNG test is performed on all approved and non-approved PRNGs (CTR DRBG, HMAC DRBG, NDRNG (Entropy), non-approved RNG (FIPS 186-2) and 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 when the `R_MODE_FILTER_FIPS140_SSL` and `R_MODE_FILTER_JCMVP_SSL` mode filters are set.
- Camellia ECB, CBC, CFB, and OFB for key sizes 128, 192, and 256 bits, which are available when the `R_MODE_FILTER_JCMVP` and `R_MODE_FILTER_JCMVP_SSL` mode filters are set.

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.

RSA signing operations implement a verification step after private key operations. This verification step, which has no effect on the signature algorithm, is in place to prevent potential faults in optimized Chinese Remainder Theorem (CRT) implementations. For more information, see https://eprint.iacr.org/2011/388.

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 22. The requirements for using the approved algorithms in a FIPS 140 mode of operation are as follows:

- Two-key Triple-DES is not allowed for encryption. Two-key Triple-DES decryption is allowed for legacy-use.
- The length of a DSA key pair for digital signature generation and verification must be either 2048 or 3072 bits. For digital signature verification, 1024 bits is allowed for legacy-use.
- The length of an RSA key pair for digital signature generation and verification must be a multiple of 512 between 2048 and 4096 bits, inclusive. For digital signature verification, a multiple of 512 greater than or equal to 1024 and less than 2048 bits is allowed for legacy-use.
- The key length for an HMAC generation or verification must be between 112 and 4096 bits, inclusive. For HMAC verification, a key length greater than or equal to 80 and less than 112 is allowed for legacy-use.
- 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, and 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 security.
- 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 security.
- When using an approved DRBG to generate keys, the requested security strength for the DRBG 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.

- For Password-based Key Derivation, the following restrictions apply:
  - Keys generated using PBKDF2 shall only be used in data storage applications.
  - The minimum password length is 14 characters, which has a strength of approximately 112 bits, assuming a randomly selected password using the extended ASCII printable character set is used.

    For random passwords (that is, a string of characters from a given set of characters in which each character is equally likely to be selected), the strength of the password is given by: `S=L*(log N/log 2)` where `N` is the number of possible characters (for example, for the ASCII printable character set `N = 95`, for the extended ASCII printable character set `N = 218`) and `L` is the number of characters. A password of the strength `S` can be guessed at random with the probability of 1 in 2^S.
  - The minimum length of the randomly-generated portion of the salt is 16 bytes.
  - The iteration count is as large as possible, with a minimum of 1000 iterations recommended.
  - The maximum key length is $(2^{32} -1)*b$, where `b` is the digest size of the message digest function.
  - Derived keys can be used as specified in NIST Special Publication 800-132, Section 5.4, options 1 and 2.

3.2 Roles

If a user of Crypto-C ME needs to operate the toolkit in different roles, then the user must ensure 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 31.
`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 31.

3.3 Modes of Operation

The following table lists and describes the available mode filters to determine the mode Crypto-C ME operates in and the algorithms allowed.

Table 6 Crypto-C ME Mode Filters

Mode	Description
`R_MODE_FILTER_FIPS140` FIPS 140-2-approved.	Implements FIPS 140-2 mode and provides the cryptographic algorithms listed in Table 4 on page 22. The default pseudo-random number generator (PRNG) is CTR DRBG.
`R_MODE_FILTER_FIPS140_SSL` FIPS 140-2-approved if used with TLS protocol implementations.	Implements FIPS 140-2 SSL mode and 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 TLS 1.0 and TLS 1.1 protocol. For more information, see Section D.2, “Acceptable Key Establishment Protocols,” in *Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program*. The implementation guidance disallows the use of the SSLv2 and SSLv3 versions. Cipher suites including non-FIPS 140-2- approved algorithms are unavailable. This mode allows implementations of the TLS protocol to operate Crypto-C ME in a FIPS 140-2-compliant manner with CTR DRBG as the default PRNG.
`R_MODE_FILTER_JCMVP` Not FIPS 140-2-approved.	Implements Japan Cryptographic Module Validation Program (JCMVP) mode and provides the cryptographic algorithms approved by the JCMVP.
`R_MODE_FILTER_JCMVP_SSL` Not FIPS 140-2-approved.	Implements JCMVP SSL mode and 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 FIPS 140-2 mode must not be shared with an application running in a non-FIPS 140-2 mode.

3.4 Operating Crypto-C ME

Crypto-C ME operates in an unrestricted mode on startup, providing access to all cryptographic algorithms available from the FIPS 140-2 provider set against the library context. To restrict the module to a specific set of algorithms, call `R_LIB_CTX_set_mode()` with one of the mode filters listed in listed in Table 6 on page 28.

After setting Crypto-C ME into a FIPS 140-2-approved mode, Crypto-C ME enforces only the algorithms listed in Table 4 on page 22 are available to operators. To disable FIPS 140-2 mode, call `R_LIB_CTX_set_mode()` with `NULL` to put Crypto-C ME back into an unrestricted mode.

`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 Operations > FIPS 140-2 Libraries” 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 5 on page 27.

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 the CTR DRBG as the default deterministic random number generator (DRNG).

Users can choose to use an approved DRNG other than the default, including the HMAC DRBG implementations, when creating a cryptographic object and setting this object against the operation requiring random number generation (for example, key generation).

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 depends on the quality of the supplied seeding (entropy). Crypto-C ME provides internal entropy collection (for example, from high precision timers) where possible. On platforms with limited internal sources of entropy, it is strongly recommended to collect entropy from external sources.

Additional entropy sources can be added to an application either by:

- Replacing internal entropy by calling `R_CR_set_info()` with `R_CR_INFO_ID_RAND_ENT_CB` and the parameters for an application-defined entropy collection callback function.
- Adding to internal entropy by calling `R_CR_entropy_resource_init()` to initialize an entropy resource structure and then adding this to the library context by calling `R_LIB_CTX_add_resource()`.

For more information about these functions, see the *RSA BSAFE Crypto-C Micro Edition Developers Guide*.

Note: If entropy from external sources is added to an application using `R_CR_set_info()` with `R_CR_INFO_ID_RAND_ENT_CB` or `R_CR_entropy_resource_init()`, no assurances are made about the minimum strength of generated keys.

For more information about seeding DRNGs, see “Randomness Recommendations for Security” in RFC 1750.

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 Developers 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_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_write_buffer_size()
BIO_should_io_special()
BIO_should_read()
BIO_should_retry()
BIO_should_small_buffer()
BIO_should_write()

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()
CRYPTOC_ME_library_info()
CRYPTOC_ME_library_version()
R_CR_add_filter()
R_CR_asym_decrypt()
R_CR_asym_decrypt_init()
R_CR_asym_encrypt()
R_CR_asym_encrypt_init()
R_CR_CTX_add_filter()
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_entropy_gather()
R_CR_entropy_resource_init()
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_memory()
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_secret_join_final()
R_CR_secret_join_init()
R_CR_secret_join_update()
R_CR_secret_split()
R_CR_secret_split_init()
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_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_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_FIPS_140_ECC_MODE_RESOURCE_LIST()
PRODUCT_FIPS_140_MODE_RESOURCE_LIST()
PRODUCT_FIPS_140_SSL_ECC_MODE_RESOURCE_LIST()
PRODUCT_FIPS_140_SSL_MODE_RESOURCE_LIST()
PRODUCT_LIBRARY_FREE()
PRODUCT_LIBRARY_INFO()
PRODUCT_LIBRARY_INFO_TYPE_FROM_STRING()
PRODUCT_LIBRARY_INFO_TYPE_TO_STRING()
PRODUCT_LIBRARY_NEW()
PRODUCT_LIBRARY_VERSION()
PRODUCT_NON_FIPS_140_MODE_RESOURCE_LIST()
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_CTX_free()
R_PASSWD_CTX_get_info()
R_PASSWD_CTX_get_passwd()
R_PASSWD_CTX_get_prompt()
R_PASSWD_CTX_get_verify_prompt()
R_PASSWD_CTX_new()
R_PASSWD_CTX_reference_inc()
R_PASSWD_CTX_set_callback()
R_PASSWD_CTX_set_info()
R_PASSWD_CTX_set_old_callback()
R_PASSWD_CTX_set_pem_callback()
R_PASSWD_CTX_set_prompt()
R_PASSWD_CTX_set_verify_prompt()
R_PASSWD_CTX_set_wrapped_callback()
R_passwd_get_cb()
R_passwd_get_passwd()
R_passwd_set_cb()
R_passwd_stdin_cb()
R_PEM_get_LIB_CTX()
R_PEM_get_PASSWD_CTX()
R_PEM_set_PASSWD_CTX()
R_PKEY_cmp()
R_PKEY_copy()
R_PKEY_CTX_add_filter()
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_generate_simple()
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_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_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_get_reason()
R_PROV_FIPS140_init_roles()
R_PROV_FIPS140_load()
R_PROV_FIPS140_load_ef()
R_PROV_FIPS140_load_env()
R_PROV_FIPS140_new()
R_PROV_FIPS140_reason_string()
R_PROV_FIPS140_ROLE_from_string()
R_PROV_FIPS140_ROLE_to_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_FIPS140_STATUS_to_string()
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_path_w()
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_fast_resource_list()
R_PROV_SOFTWARE_get_default_small_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_new()
R_SELECT_set_info()
R_SKEY_delete()
R_SKEY_dup()
R_SKEY_dup_ef()
R_SKEY_free()
R_SKEY_generate()
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_disable_cpu_features()
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_data()
STACK_delete()
STACK_delete_all()
STACK_delete_all_arg()
STACK_delete_ptr()
STACK_dup()
STACK_dup_ef()
STACK_find()
STACK_for_each()
STACK_free()
STACK_insert()
STACK_lfind()
STACK_move()
STACK_new()
STACK_new_ef()
STACK_num()
STACK_pop()
STACK_pop_free()
STACK_pop_free_arg()
STACK_push()
STACK_set()
STACK_set_cmp_func()
STACK_shift()
STACK_unshift()
STACK_value()
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_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.
BPS	Brier, Peyrin and Stern. An encryption mode of operation used with the AES and Triple DES symmetric key algorithms for format-preserving encryption (FPE).
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 producing a stream of ciphertext bits rather than a succession of blocks. In other respects, it has similar properties to the CBC mode of operation.
CMVP	Cryptographic Module Validation Program
CRNG	Continuous Random Number Generation.
CTR	Counter mode of encryption, which turns a block cipher into a stream cipher. It generates the next keystream block by encrypting successive values of a counter.
CTR DRBG	Counter mode Deterministic Random Bit Generator.
CTS	Cipher text stealing mode of encryption, which enables block ciphers to be used to process data 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.
DESX	A variant of the DES symmetric key algorithm intended to increase the complexity of a brute force attack.

Table 7 Acronyms and Definitions

Term	Definition
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 comprising 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.
ECB	Electronic Codebook. A mode of encryption, which 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 and decrypts (undoes the encryption) the ciphertext.
FIPS	Federal Information Processing Standards.
FPE	Format-preserving encryption. Encryption where the ciphertext output is in the same format as the plaintext input. For example, encrypting a 16-digit credit card number produces another 16-digit number.
GCM	Galois/Counter Mode. A mode of encryption combining the Counter mode of encryption with Galois field multiplication for authentication.
GMAC	Galois Message Authentication Code. An authentication only variant of GCM.
GOST	GOST symmetric key encryption algorithm developed by the USSR government. There is also the GOST message digest algorithm.
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.

Table 7 Acronyms and Definitions

Term	Definition
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, which hashes an arbitrary-length input into a 16-byte digest. MD2 is no longer considered secure.
MD4	A message digest algorithm, which hashes an arbitrary-length input into a 16-byte digest.
MD5	A message digest algorithm, which 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, which 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, which 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.

Table 7 Acronyms and Definitions

Term	Definition
RNG	Random Number Generator.
RSA	Public key (asymmetric) algorithm providing the ability to encrypt data and create and verify digital signatures. RSA stands for Rivest, Shamir, and Adleman, the developers of the RSA public key cryptosystem.
SEED	SEED symmetric key encryption algorithm developed by the Korean Information Security Agency.
SHA	Secure Hash Algorithm. An algorithm, which creates a unique hash value for each possible input. SHA takes an arbitrary input, which 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, which 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, SHA-512, SHA-512/224, and SHA-512/256), which produce digests of 224, 256, 384, 512, 224, and 256 bits respectively.
SEED	A symmetric key algorithm developed by the Korean Information Security Agency.
Triple DES	A variant of DES, which uses three 56-bit keys.
XTS	XEX-based Tweaked Codebook mode with ciphertext stealing. A mode of encryption used with AES.