RSA® BSAFE® Crypto-C Micro Edition 4.1.5 Security Policy Level 1 with Level 2 Roles, Services and Authentication

This document is a non-proprietary Security Policy for the RSA BSAFE Crypto-C Micro Edition 4.1.5 (Crypto-C ME) cryptographic module from Dell Australia Pty Limited, BSAFE Product Team.

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

Contents:

Preface

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

Federal Information Processing Standards Publication 140-2 - Security Requirements for Cryptographic Modules (FIPS 140-2) 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 website.

References

This document deals only with operations and capabilities of the Crypto-C ME cryptographic module in the technical terms of a FIPS 140-2 cryptographic module security policy. More information about Crypto-C ME and the entire Dell BSAFE product line is available at Dell Support.

Document Organization

This Security Policy explains the cryptographic module features and functionality relevant to FIPS 140-2, and comprises the following sections:

- This section, provides an overview and introduction to the Security Policy.
- The Cryptographic Module describes Crypto-C ME and how it meets FIPS 140-2 requirements.
- Secure Operation of the Module specifically addresses the required configuration for the FIPS 140-2 mode of operation.
- Services lists the functions of Crypto-C ME.
- Acronyms and Definitions lists the acronyms and definitions used in this document.

Terminology

In this document, the terms *cryptographic module* and *module*, refer to the Crypto-C ME FIPS 140-2 validated cryptographic module for Overall Security Level 1 with Level 2 Roles, Services and Authentication, and Level 3 Design Assurance.

1 The Cryptographic Module

Crypto-C ME is designed for different processors, and includes various optimizations. Assembly-level optimizations on key processors mean Crypto-C ME algorithms can be used at increased speeds on many platforms.

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

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.

This section provides an overview of the cryptographic module and contains the following topics:

- Laboratory Validated Operating Environments
- Affirmation of Compliance for other Operating Environments
- Module Characteristics
- Module Interfaces
- Roles, Services and Authentication
- Cryptographic Key Management
- Cryptographic Algorithms
- Self Tests.

1.1 Laboratory Validated Operating Environments

For FIPS 140-2 validation, Crypto-C ME is tested by an accredited FIPS 140-2 testing laboratory. The referenced platforms were tested both with and without the Processor Algorithm Accelerators (PAA). Refer to Table 1 for details.

Validation testing is completed on the following operating environments:

- Apple®:
  - iOS® 12 on ARM®v8 (64-bit) running on an iPhone® 8 with an Apple A11 processor (PAA 1) built with Xcode® 9
  - macOS® 10.15 on x86_64 (64-bit) running on VMware ESXi™ 6.7.0 on a Mac Pro® with an Intel® Xeon® E5-1650 v2 processor (PAA 2) built with Xcode 7.3.
- Canonical® Ubuntu® 16.04 Long Term Support (LTS) on ARMv7 (32-bit) running on a BeagleBoard.org® BeagleBone® Black with a Texas Instruments™ Sitara® AM335x processor built with gcc 8.4 (hard float).
- FreeBSD® Foundation, FreeBSD 11.3 on x86_64 (64-bit) running on VMware ESXi 6.7.0 on a Dell™ PowerEdge R640 with an Intel Xeon Gold 6136 processor (PAA 2) built with Clang 8.0.
- Google ®Android ®10.0 on:
  - ARMv8 (64-bit) running on a Pixel™ 3 with Qualcomm® Snapdragon™ 845 (PAA 1) built with Android SDK 21 with Clang 9
  - ARMv7 (32-bit) running on a Pixel 3 with Qualcomm Snapdragon 845 built with Android SDK 21 with Clang 9.
- IBM AIX® 7.2 on:
  - PowerPC® (64-bit) running on PowerVM® Virtual I/O Server 2.2.6.41 on an IBM Power® 8284-22A with an IBM POWER8 ®processor built with XL C/C++ for AIX (XLC) v11.1
  - PowerPC (32-bit) running on PowerVM Virtual I/O Server 2.2.6.41 on an IBM Power 8284-22A with an IBM POWER8 processor built with XLC v11.1.
- Microsoft®:
  - Windows® 10 Enterprise on:
    - x86_64 (64-bit) running on VMware ESXi 6.7.0 on a Dell PowerEdge R640 with Intel Xeon Gold 6136 processor (PAA 2) built with Visual Studio® 2017
    - x86 (32-bit) running on VMware ESXi 6.7.0 on a Dell PowerEdge R640 with Intel Xeon Gold 6136 processor (PAA 2) built with Visual Studio 2017
    - x86_64 (64-bit) running on VMware ESXi 6.7.0 on a Dell PowerEdge R640 with Intel Xeon Gold 6136 processor (PAA 2) built with Visual Studio 2013.
  - Windows Server® 2019 on x86_64 (64-bit) running on:
    - VMware ESXi 6.7.0 on a Dell PowerEdge R640 with Intel Xeon Gold 6136 processor (PAA 2) built with Visual Studio 2017

        - VMware ESXi 6.7.0 on a Dell PowerEdge R7425 with AMD™ EPYC™ 7451 processor (PAA 3) built with Visual Studio 2017.
    - Windows Server 2016 on x86_64 (64-bit) running on VMware ESXi 6.7.0 on a Dell PowerEdge R640 with Intel Xeon Gold 6136 processor (PAA 2) built with Visual Studio 2017.
- Oracle® Solaris ®11.4 on:
    - SPARC® v9 (64-bit) running on VM Server for SPARC 11, with a SPARC T4-2 processor (PAA 4) built with Sun C 5.13
    - SPARC v8+ (32-bit) running on VM Server for SPARC 11, with a SPARC T4-2 processor (PAA 4) built with Sun C 5.13
    - x86_64 (64-bit) running on VMware ESXi 6.7.0 on a Dell PowerEdge R640 with Intel Xeon Gold 6136 processor (PAA 2) built with Sun C 5.13.
- Red Hat® Enterprise Linux® 7.8 on PowerPC (64-bit) running on PowerVM Virtual I/O Server 2.2.6.41 on an IBM Power 8284-22A with an IBM POWER8 processor built with gcc 4.4.
- SUSE Software Solutions®:
    - SUSE® Linux Enterprise Server 12 SP5 on:
        - PowerPC (64-bit) running on PowerVM Virtual I/O Server 2.2.6.41 on an IBM Power 8284-22A with an IBM POWER8 processor built with gcc 8.3
        - x86_64 (64-bit) running on VMware ESXi 6.7.0 on a Dell PowerEdge R640 with Intel Xeon Gold 6136 processor (PAA 2) built with gcc 8.3
        - x86 (32-bit) running on VMware ESXi 6.7.0 on a Dell PowerEdge R640 with Intel Xeon Gold 6136 processor (PAA 2) built with gcc 8.3
        - ARMv8 (64-bit) running on a SoftIron® Overdrive 1000 with an AMD Opteron™ A1100 processor (PAA 1) built with gcc 8.2.
    - SUSE Linux Enterprise Server 11 SP4 LTSS on PowerPC (64-bit) running on PowerVM Virtual I/O Server 2.2.6.41 on an IBM Power 8284-22A with an IBM POWER8 processor built with gcc 4.4.

Table 1 Processor Algorithm Accelerator Testing

Reference	Processor	PAA	Algorithms
1	ARMv8 (64-bit)	NEON™ and Cryptography Extensions	AES and SHA
2	Intel x86 (32-bit), x86_64 (64-bit)	AES-NI	AES
3	AMD x86_64 (64-bit)	AES-NI and SHA Extensions	AES and SHA
4	Oracle SPARC T series	SPARC	AES, DES and SHA

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 if the specific operational environment is not listed on the validation certificate.

> **Important:** Dell affirms compliance of all patch and Service Pack levels with the same capabilities as the listed operating environments, unless noted otherwise.

For Crypto-C ME 4.1.5, Dell affirms compliance for the following operating environments:

- Apple:
  - macOS 10.14 on:
    - 86_64 (64-bit)
    - x86 (32-bit).
  - macOS 10.13 on:
    - x86_64 (64-bit)
    - x86 (32-bit).
- Canonical:
  - Ubuntu 20.04 LTS on:
    - x86_64 (64-bit)
    - x86 (32-bit).
  - Ubuntu 18.04 LTS on:
    - x86_64 (64-bit)
    - x86 (32-bit).
  - Ubuntu 16.04 LTS on:
    - x86_64 (64-bit)
    - x86 (32-bit).
- CentOS™ Project:
  - CentOS 8.0 on:
    - x86_64 (64-bit)
    - x86 (32-bit).
  - CentOS 7.8 on:
    - x86_64 (64-bit)
    - x86 (32-bit).

  - CentOS 6.10 on:
    - x86_64 (64-bit)
    - x86 (32-bit).
- Dell PowerProtect™ Data Domain™ OS on x86_64 (64-bit).
- FreeBSD® Foundation FreeBSD 12.1 on x86_64 (64-bit).
- Google:
  - Android 9.0 on ARM v8 (64-bit)
  - Android 8.0 on ARM v8 (64-bit)
  - Android 7.1.1 on ARM v8 (64-bit).
- HPE
  - HP-UX 11.31 on:
    - Itanium2 64-bit
    - Itanium2 32-bit
    - 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.
- IBM:
  - AIX v7.1 on:
    - PowerPC (64-bit)
    - PowerPC (32-bit).
- Microsoft:
  - Windows 10 Enterprise on:
    - x86 (32-bit), built with Visual Studio 2013.
  - Windows 10 IoT Enterprise LTSC 2019 on:
    - x86_64 (64-bit), built with Visual Studio 2017
    - x86 (32-bit), built with Visual Studio 2017.
  - Windows 8.1 Enterprise on:
    - x86_64 (64-bit), built with Visual Studio 2017
    - x86_64 (64-bit), built with Visual Studio 2013
    - x86_64 (64-bit), built with Visual Studio 2010
    - x86 (32-bit), built with Visual Studio 2017
    - x86 (32-bit), built with Visual Studio 2013.
  - Windows Server 2012 Standard on:
    - x86_64 (64-bit), built with Visual Studio 2017
    - x86_64 (64-bit), built with Visual Studio 2013
    - x86_64 (64-bit), built with Visual Studio 2010.

  - Windows Server 2012 R2 Standard on:
    - x86_64 (64-bit), built with Visual Studio 2017
    - x86_64 (64-bit), built with Visual Studio 2013
    - x86_64 (64-bit), built with Visual Studio 2010.
- Oracle:
  - Linux 8 on:
    - ARMv8 (64-bit)
    - x86_64 (64-bit)
  - Linux 7 on:
    - ARMv8 (64-bit)
    - x86_64 (64-bit)
  - Solaris 11.4 on:
    - SPARC v8 (32-bit), built with Sun C 5.13.
  - Solaris 10 Update 11 on:
    - SPARC v9-T4 (64-bit)
    - SPARC v9-T2 (64-bit)
    - SPARC v8+ (32-bit)
    - SPARC v8 (32-bit)
    - x86_64 (64-bit)
    - x86 (32-bit)
- Red Hat:
  - Enterprise Linux 8.1 on:
    - x86_64 (64-bit)
    - x86 (32-bit)
    - PowerPC (64-bit)
  - Enterprise Linux 7.8 on
    - x86_64 (64-bit)
    - x86 (32-bit)
    - PowerPC (32-bit)
    - IBM S/390x (64-bit)
    - IBM S/390 (31-bit)
  - Enterprise Linux 7.6 on:
    - PowerPC (64-bit)
    - PowerPC (32-bit)

  - Enterprise Linux 7.4 on ARMv8 (64-bit)
  - Enterprise Linux 6.10 on:
    - x86_64 (64-bit)
    - x86 (32-bit).
- SUSE Software Solutions®:
  - SUSE® Linux Enterprise Server 15 SP4 on:
    - x86_64 (64-bit)
  - SUSE® Linux Enterprise Server 15 SP2 on:
    - x86_64 (64-bit)
    - PowerPC (64-bit)
  - SUSE Linux Enterprise Server 15 SP1 on:
    - x86_64 (64-bit)
    - x86 (32-bit)
    - PowerPC (64-bit)
  - SUSE Linux Enterprise Server 15 on:
    - x86_64 (64-bit)
    - x86 (32-bit)
    - PowerPC (64-bit)
  - SUSE Linux Enterprise Server 12 SP4 and SP2 on:
    - ARMv8 (64-bit)
    - x86_64 (64-bit)
    - x86 (32-bit)
    - PowerPC (64-bit).
  - SUSE Linux Enterprise Server 12 SP3 on:
    - ARMv8 (64-bit)
    - x86_64 (64-bit)
    - x86 (32-bit)
    - PowerPC (64-bit)
    - IBM S/390x (64-bit)
    - IBM S/390 (31-bit)
  - SUSE Linux Enterprise Server 11 SP4 LTSS on:
    - Itanium 2 (64-bit)
    - Power PC (32-bit)
    - x86_64 (64-bit)
    - x86 (32-bit)

1.3 Module Characteristics

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 Security Level 2 for Roles, Services and Authentication, Security Level 3 for Design Assurance, and Overall Security Level 1 security requirements.

Crypto-C ME is packaged as a set of dynamically loaded shared libraries containing the module's entire executable code. The Crypto-C ME toolkit relies on the physical security provided by the hosting general purpose computer (GPC) in which it runs. A Level 2 hosting GPC operational environment should incorporate a Common Criteria Evaluation Assurance Level 2 (EAL2) operating system and the enclosure should be at least opaque and be either lockable or tamper evident.

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

Table 2 Certification Levels

Section of the FIPS 140-2 Specification	Level
Cryptographic Module Specification	3
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

1.3.1 Single Operator Mode

An Operator is an individual accessing the cryptographic module or a process operating the cryptographic module on behalf of the individual.

The operating system must enforce a single operator mode of operation, that is, concurrent operators are explicitly excluded.

Single-user Operating Systems

The following supported operating systems are single-user operating systems, so no steps are required to configure a single operator mode of operation:

- Apple iOS
- Google Android.

Multi-user Operating Systems

For the following supported multi-user operating systems, the operating system and hardware enforce a single operator mode of operation by enforcing process isolation and CPU scheduling:

- Apple macOS
- Canonical Ubuntu
- CentOS Project CentOS
- Dell PowerProtect
- FreeBSD Foundation FreeBSD
- Google Android
- HPE HP-UX
- IBM AIX
- Microsoft Windows
- Oracle Solaris
- Red Hat Enterprise Linux
- SUSE Software Solutions SUSE.

On these operating systems, running on a general purpose computer, dynamically loaded shared libraries, including the cryptographic module, are loaded into the address space of a process. Each instance of the cryptographic module functions entirely within the process space of the process containing the module.

The single operator for a given instance of the cryptographic module is the identity associated with the process containing the module. The operating system and hardware enforce process isolation including memory, where keys and intermediate key data are stored, and CPU scheduling. The writable memory areas of the cryptographic module, data and stack segments, are accessible only to the process containing the module.

The operating system is responsible for multitasking operations so that other processes cannot access the address space of the process containing the cryptographic module. Consequently, with the exception of privileged user accounts, no additional steps are required to restrict the operating system to a single operator mode of operation. That is, concurrent operators are explicitly excluded.

Privileged user accounts

Multi-user operating systems provide tracing and debugging utilities through which one process can control another, enabling the controller process to inspect and manipulate the internal state of its target process.

With the exception of privileged user accounts, root user/administrator user, the controller process must be running as the same user id as the target process for these utilities to work. This usage does not contravene the single operator mode of operation as both the controller and target processes are operating on behalf of a single operator.

Privileged user accounts are able to use tracing and debugging utilities to target a process with a different user id to the controlling process. An operator using this privilege to inspect or manipulate a process operating on behalf of another operator contravenes the single operator mode of operation.

To maintain the single operator mode of operation a privileged user must not use any of the system tracing and debugging utilities provided by the operating system.

- In Unix-type operating systems the `ptrace` system call, the debugger `gdb`, `strace`, `ftrace` and `systemtrap` must not be used.
- On Windows equivalent system tracing and debugging utilities must not be used.

If necessary, the operating system can be configured to provide only a single operator. That is, login credentials for all user accounts, including privileged user accounts, can be provided to a single individual only.

Server environments

When the module is deployed in a server environment, the server application is the user of the module. The server application makes the calls to the module. Therefore, the server application is the single user of the module, even when the server application is serving multiple clients.

1.4 Module Interfaces

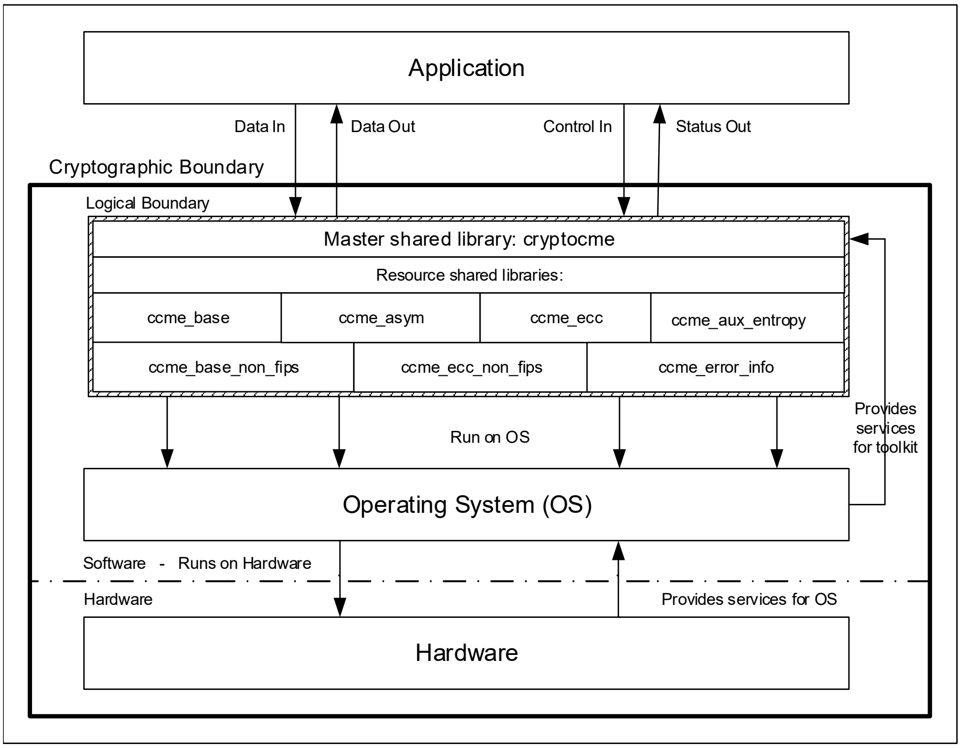
Crypto-C ME is validated as a multi-chip standalone cryptographic module. The physical cryptographic boundary of the module is the case of the general-purpose computer or mobile device, which encloses the hardware running the module. The physical interfaces for Crypto-C ME consist of the keyboard, mouse, 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 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 operating system
  - `libcryptocme.sl` on systems running an HP-UX operating system
  - `libcryptocme.dylib` on systems running an Apple 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`, 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`, 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`, 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`, and `libccme_error_info.dylib` on systems running an Apple operating system.

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 in 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: For systems running an Apple or Windows operating system, the logical boundary of the shared libraries includes only the library code and data sections, and does not include other shared library file content, such as any code signatures.

1.5 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.

1.5.1 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 using 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, which is stored in the roles database.

The application must set random PIN data of sufficient security strength to make a brute force attack infeasible. The PIN must contain random data with the equivalent of a minimum of 73 effectively random bits. Reasoning for this figure is provided in Roles Database Authentication PIN Threat Model:

Up to 64 bytes of PIN data can be passed to the module by a call to assume a role.

1.5.2 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:

- Create the roles database, in memory or on disk
- Perform the full set of self tests.
- Call any Crypto-C ME function. For a complete list of functions available to the Crypto Officer, see Services.

1.5.3 Crypto User Role

A Crypto Officer 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_tests_full()`, which is reserved for the Crypto Officer. For a complete list of Crypto-C ME functions, see Services.

1.5.4 Unloading and Reloading the Module

A roles database 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 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 using 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.

1.6 Cryptographic Key Management

Cryptographic key management is concerned with generating keys, key assurance, storing keys, managing access to keys, protecting keys during use, and zeroizing keys when they are no longer required.

1.6.1 Key Generation

Crypto-C ME supports the generation of DSA, RSA, Diffie-Hellman (DH) and Elliptic Curve Cryptography (ECC) public and private keys. 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.

When operating in a FIPS 140-2-approved manner, RSA keys can only be generated using the approved FIPS 186-4 RSA key generation method.

1.6.2 Key Assurance

Crypto-C ME supports validity assurance of asymmetric keys. Functions are available to test the validity of:

- ECC keys, and DSA keys and domain parameters, against FIPS 186-4
- ECC keys, and DH keys and domain parameters, against SP 800-56A Rev. 3
- RSA keys against FIPS 186-4 or SP 800-56B Rev. 2.

1.6.3 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 Critical Security Parameters (CSPs) in the module and where they are stored.

Table 3 Key Storage

Key or CSP	**Generation/Input/Output**	**Storage**
Hardcoded DSA public key	• Generated when the module is created • Cannot be output from the module.	Persistent storage embedded in the module binary
Hardcoded HMAC key (128-bit) to check integrity of the authentication file	• Generated when the module is created. • Cannot output the key.	Volatile memory only (plaintext)
AES keys	• Entered in plaintext through the API or generated by an explicit API call • Output in plaintext through the API.	Volatile memory only (plaintext)
HMAC keys	• Entered in plaintext through the API or generated by an explicit API call • Output in plaintext through the API.	Volatile memory only (plaintext)

Table 3 Key Storage (continued)

Key or CSP	Generation/Input/Output	Storage
DH public/private keys	• Entered in plaintext through the API or generated by an explicit API call • Output in plaintext through the API.	Volatile memory only (plaintext)
ECC public/private keys	• Entered in plaintext through the API or generated by an explicit API call • Output in plaintext through the API.	Volatile memory only (plaintext)
RSA public/private keys	• Entered in plaintext through the API or generated by an explicit API call • Output in plaintext through the API.	Volatile memory only (plaintext)
DSA public/private keys	• Entered in plaintext through the API or generated by an explicit API call • Output in plaintext through the API.	Volatile memory only (plaintext)
CTR DRBG entropy	• Generated internally • Cannot be output from the module.	Volatile memory only (plaintext)
CTR DRBG V value	• Generated internally • Cannot be output from the module.	Volatile memory only (plaintext)
CTR DRBG key	• Generated internally • Cannot be output from the module.	Volatile memory only (plaintext)
CTR DRBG init_seed	• Generated internally • Cannot be output from the module.	Volatile memory only (plaintext)
HMAC DRBG entropy	• Generated internally • Cannot be output from the module.	Volatile memory only (plaintext)
HMAC DRBG V value	• Generated internally • Cannot be output from the module.	Volatile memory only (plaintext)
HMAC DRBG key	• Generated internally • Cannot be output from the module.	Volatile memory only (plaintext)
HMAC DRBG init_seed	• Generated internally • Cannot be output from the module.	Volatile memory only (plaintext)
Role-based authentication token	• Entered in plaintext through the API. • Cannot output the CSP.	Volatile memory only (plaintext)

CSP Usage:

- The hardcoded DSA public key is used to confirm the integrity of the module binaries during the module integrity POST.
- The hardcoded HMAC key (128-bit) is used to confirm the integrity of the module file that contains the current role-based authentication tokens.
- The DRBG CSPs (V value, key, init_seed and entropy) are all required for the correct operation of DRBG instances, as per SP 800-90A. The V value and the key represent the internal state of the DRBG. The init_seed is entropic data that is used to initialize the internal state of the DRBG.

- The role-based authentication token confirms the application has access to a chosen role.
- All other CSPs are loaded or generated by application calls to the module and are used in cryptographic operations performed by the application.

1.6.4 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 4 Key and CSP Access

Service Type	Key or CSP	Type of Access
Asymmetric encryption and decryption	Asymmetric keys (RSA)	Read/Execute
Symmetric encryption and decryption	Symmetric keys (AES)	Read/Execute
Digital signature and verification	Asymmetric keys (DSA, ECC, 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, IV, key, and init_seed	Read/Write/Execute
Key derivation	Symmetric Keys (AES) MAC Keys (HMAC)	Write
Key generation	Symmetric keys (AES) Asymmetric keys (DSA, RSA, DH, and ECC) MAC keys (HMAC)	Write
Key assurance	Asymmetric keys (DSA, RSA, DH and ECC)	Read
Key establishment primitives	Asymmetric keys (RSA, DH, ECC)	Read/Execute
Role-based authentication token	PIN	Read/Write
Self-test (Crypto Officer service)	Hardcoded DSA public key	Read/Execute
Show status	None	N/A
Zeroization	All	Read/Write

1.6.5 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.

1.6.6 Key Wrapping

Crypto-C ME supports wrapping of raw key data, symmetric keys, and asymmetric keys with:

- Symmetric keys - AES KW and AES KWP algorithms.
- Asymmetric keys - RSA-OAEP algorithm.

1.7 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.

Note: Crypto User Guidance on Algorithms provides algorithm-specific guidance on the use of the algorithms listed in this section.

1.7.1 FIPS 140-2-approved Algorithms

The following table lists the Crypto-C ME FIPS 140-2-approved algorithms, with appropriate standards and CAVP validation certificate numbers:

Table 5 Crypto-C ME FIPS 140-2-approved Algorithms

Algorithm Type	Algorithm and approved parameter/modulus/key sizes	Standard	Validation Certificate
Asymmetric Cipher	RSADP (RSA decryption primitive) component Modulus sizes: 2048 and 3072[1] bits	SP 800-56B Rev. 2	C2130
	RSAEP (RSA encryption primitive) component Modulus sizes: 2048 and 3072 bits	SP 800-56B Rev. 2	VA[2]
Asymmetric Key	ECC • Public Key Validation Curves: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	SP 800-56A Rev. 3[3]	VA
	• Key Pair Generation Curves: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	FIPS 186-4	C2130
	FFC • Domain Parameter Generation L = 2048, N = 224; L = 2048, N = 256; L = 3072, N = 256	FIPS 186-4	C2130
	• Domain Parameter Validation L = 1024, N = 160	FIPS 186-2	C2130
	• Domain Parameter Validation L = 1024, N = 160; L = 2048, N = 224; L = 2048, N = 256; L = 3072, N = 256	FIPS 186-4	C2130
	• Key Pair Generation L = 2048, N = 224; L = 2048, N = 256; L = 3072, N = 256	FIPS 186-4	C2130
	• Key Pair Validation L = 2048, N = 224; L = 2048, N = 256; L = 3072, N = 256	SP 800-56A Rev. 3[3]	VA
	RSA • Key Generation, Modulus sizes: 2048, 3072 bits	FIPS 186-4	C2130
	• Key Validation, Modulus sizes: 2048 bits and larger	SP 800-56B Rev. 2	VA

Table 5 Crypto-C ME FIPS 140-2-approved Algorithms (continued)

Algorithm Type	Algorithm and approved parameter/modulus/key sizes	Standard	Validation Certificate
Digital Signature	DSA • Signature Generation L = 2048, N = 224; L = 2048, N = 256; L = 3072, N = 256 • Signature Verification L = 1024, N = 160; L = 2048, N = 224; L = 2048, N = 256; L = 3072, N = 256	FIPS 186-4 FIPS 186-4	C2130
	ECDSA • Signature and Signature Component Generation Curves: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 • Signature Verification Curves: B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521	FIPS 186-4 FIPS 186-4	C2130
	RSA • Signature Generation Algorithms: X9.31, PKCS #1 V1.5, RSASSA-PSS Key (modulus) sizes: 2048, 3072 bits. • Signature Verification Algorithms: X9.31, PKCS #1 V1.5, RSASSA-PSS Key (modulus) sizes: 1024, 2048, 3072 bits. • Signature Verification Algorithms: X9.31, PKCS #1 V1.5, RSASSA-PSS Key (modulus) sizes: 1024, 1536, 2048, 3072, 4096 bits. • RSASP1 (RSA signature primitive 1) component Key (modulus) sizes: 2048 and 3072 [1] bits	FIPS 186-4 FIPS 186-4 FIPS 186-2 FIPS 186-4	C2130
Key Agreement Primitives	FFC • Domain parameter-size sets: L=2048, N=224; L=2048, N=256 • Approved IKE groups: MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 • Approved TLS groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192	SP 800-56A Rev. 3[3]	VA
Key Agreement Schemes	KAS-SSC ECC • Schemes: Full Unified Model, Ephemeral Unified Model, One-Pass Unified Model, One-Pass Diffie-Hellman Model and Static Unified Model • Curves: P-224, P-256, P-384, P-521	SP 800-56A Rev. 3[3]	VA
	KAS-SSC FFC • Schemes: dhHybrid1, dhEphem, dhHybridOneFlow, dhOneFlow and dhStatic • Domain parameter-size sets: L=2048, N=224; L=2048, N=256	SP 800-56A Rev. 3[3]	VA

Table 5 Crypto-C ME FIPS 140-2-approved Algorithms (continued)

Algorithm Type	Algorithm and approved parameter/modulus/key sizes	Standard	Validation Certificate
Key Derivation Functions (KDFs)	HMAC-based Extract-and-Expand KDF (HKDF): SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	SP 800-56C Rev. 1	VA
	Key-based KDF (KBKDF), using pseudo-random functions: HMAC-based Feedback Mode[4], with: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	SP 800-108	C2130
	Password-based KDF 2 (PBKDF2) [5]	SP 800-132	VA[6]
	Single-step KDF	SP 800-56C Rev. 1	VA
	SSH KDF: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	SP 800-135 Rev. 1	C2130
	TLS KDF: TLS 1.0/1.1[7] TLS 1.2: SHA-256, SHA-384, SHA-512	SP 800-135 Rev. 1	C2130
	X9.63 KDF - Component Test: SHA-224, SHA-256, SHA-384, SHA-512	ANSI X9.63, SP 800-135 Rev. 1	C2130
Key Generation	Cryptographic Key Generation (CKG)	SP 800-133 Rev. 2	VA
Key Transport Schemes	KTS-OAEP, KTS-OAEP-Party_V-confirmation. Modulus sizes: 2048 bits and larger	SP 800-56B Rev. 2	VA
Key Wrap	AES in KW and KWP modes with 128, 192, and 256-bit key sizes	SP 800-38F	C2130
	RSA-OAEP Modulus sizes: 2048 bits and larger	SP 800-56B Rev. 2	VA as part of Key Transport Schemes[6]
MAC	GMAC: AES-128, AES-192, AES-256	SP 800-38D	C2130
	HMAC SHA: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256	FIPS 198-1	C2130
	HMAC SHA-3: SHA3-224, SHA3-256, SHA3-384, SHA3-512	FIPS 202	C2130

Table 5 Crypto-C ME FIPS 140-2-approved Algorithms (continued)

Algorithm Type	Algorithm and approved parameter/modulus/key sizes	Standard	Validation Certificate
Message Digest	SHA: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256	FIPS 180-4	C2130
	SHA-3: SHA3-224, SHA3-256, SHA3-384, SHA3-512	FIPS 202	C2130
Random Bit Generator	CTR DRBG AES-CTR mode with 128, 192, and 256-bit key sizes.	SP 800-90A Rev. 1	C2130
	HMAC DRBG Modes SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256	SP 800-90A Rev. 1	C2130
	SHA3-224, SHA3-256, SHA3-384, SHA3-512	FIPS 202	VA
Symmetric Cipher	AES CBC, CFB 128-bit, ECB, OFB 128-bit, and CTR modes with 128, 192, and 256-bit key sizes CCM modes with 128, 192, and 256-bit key sizes GCM mode with automatic internally generated IV with 128, 192, and 256-bit key sizes XTS mode with 128 and 256-bit key sizes.	SP 800-38A SP 800-38C SP 800-38D SP 800-38E	C2130

[1] A 3072-bit modulus is not tested by the CAVP but is approved for use in the FIPS 140-2 approved mode of operation. Dell affirms correct implementation of RSADP and RSASP1 with a 3072-bit modulus.

[2] Vendor Affirmed.

[3] Dell affirms compliance with SP 800-56A Rev. 3 as detailed in IG D.1-rev3.

[4] As defined by the HKDF expand step,

[5] As defined in SP 800-132, PBKDF2 can be used in FIPS 140-2 approved mode of operation when used with FIPS 140-2-approved symmetric key and message digest algorithms. For more information, see Crypto User Guidance.

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

[7] The TLS 1.0 and 1.1 KDF, documented in SP 800-135, are only allowed when the conditions detailed in the Crypto User Guidance are satisfied.

1.7.2 FIPS 140-2-allowed Algorithms

The following table lists the Crypto-C ME FIPS 140-2-allowed algorithms, with appropriate standards:

Table 6 Crypto-C ME FIPS 140-2-allowed Algorithms

Algorithm Type	Algorithm	Standard
Message Digest	MD5[1] • As part of an approved key transport scheme, for example, TLS 1.0, where no security is provided by the MD5 algorithm.	SP 800-135 Rev. 1 RFC 2246 RFC 4346
Random Number	Non-deterministic Random Number Generator (NDRNG) Entropy source to seed the random number generator.	IG G.13

[1] MD5 is allowed in FIPS140-2 approved mode of operation for a purpose that is not security relevant or is redundant to an approved cryptographic algorithm. See section 4.2.1 of SP 800-135 Rev. 1 and IG 1.23

1.7.3 Non-FIPS 140-2-approved Algorithms

The following table lists the algorithms that are not FIPS 140-2-approved:

Table 7 Crypto-C ME non-FIPS 140-2-approved Algorithms

Algorithm Type	Algorithm
Asymmetric Key	ECIES, DH
Key Agreement Primitives	ECC, FFC
Key Derivation Function	SCrypt PBKDF1 Shamir's Secret Share
Key Encapsulation	RSA PKCS #1 v1.5 key decryption Modulus sizes: 2048 to 15360 in increments of 256 bits
Key Transport Schemes	KTS-KEM-KWS, KTS-KEM-KWS-Party_V-confirmation. Modulus sizes: 2048 bits and larger
Key Wrap	RSA-KEM-KWS Modulus sizes: 2048 bits and larger
Message Authentication Code	HMAC-MD5
Message Digest	MD2, MD4
Random Number	Non-approved RNG (FIPS 186-2) Non-approved RNG (OTP).

Table 7 Crypto-C ME non-FIPS 140-2-approved Algorithms (continued)

Algorithm Type	Algorithm
Symmetric Cipher	AES in CFB 64-bit, CBC-CS3 (CTS), and BPS[1] modes ARIA DES, Triple-DES (two-key), DESX, DES40, DES in BPS mode Camellia GOST RC2, RC4, RC5 SEED Triple-DES (three key), CBC, CFB 64-bit, ECB, and OFB 64-bit modes

[1] For format-preserving encryption (FPE).

For more information about using Crypto-C ME in a FIPS 140-2-compliant manner, see Secure Operation of the Module.

1.8 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.

1.8.1 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)
- RSA KATs (encrypt/decrypt)
- SHA-1,
  SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256,
  SHA3-224, SHA3-256, SHA3-384, and SHA3-512 KATs
- HMAC SHA-1,
  HMAC SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, HMAC
  SHA3-224, SHA3-256, SHA3-384, and SHA3-512 KATs
- ANSI X9.63 KDF
  HKDF
  Single-step KDF
  SSH KDF
  TLS 1.0/1.1 KDF
  TLS 1.2 KDF KATs
- RSA sign/verify KATs
- RSA sign/verify test
- DSA sign/verify test
- ECDSA sign/verify test
- DH, ECDH and ECDHC pair-wise consistency tests
- PRNG (CTR DRBG and HMAC DRBG) KATs
- Software integrity test using DSA signature verification.

Power-up self-tests are executed automatically when the module loads into memory.

1.8.2 Conditional Self-tests

Crypto-C ME performs two conditional self-tests:

- A pair-wise consistency test each time Crypto-C ME generates a DH, DSA, RSA, or ECC 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 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)).
- DRBG health tests are run during instantiation, random generation, and re-seeding by the toolkit.

1.8.3 Mitigation of Other Attacks

The following table describes the mechanisms employed to mitigate against attacks which might prevent proper operation of the module.

Table 8 Mitigation of Other Attacks

Attack	Mitigation Mechanism
Side-channel attacks on RSA and ECC private key operations.	Blinding
Fault attack on RSA-CRT	Verify after Sign
Padding Oracle Attack on PKCS #1	Constant time padding operation

Blinding:

RSA and ECC private key operations implement blinding, a reversible way of modifying the input data so as to make the operation immune to timing attacks. Blinding has no effect on the algorithms other than to mitigate side-channel attacks on the algorithm. Blinding is enabled by default but can be turned off for performance reasons in situations where timing attacks are not possible.

Verify after sign:

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 Modulus Fault Attacks Against RSA-CRT Signatures.

Constant time padding operation:

RSA PKCS#1 v1.5 encryption padding operations are implemented in constant time in order to make the operation immune to timing attacks. For more information, see Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1.

2 Secure Operation of the Module

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

> **Note:** The module operates as a Validated Cryptographic Module only when the rules for secure operation are followed.

2.1 Crypto User Guidance

This section provides guidance to the module user to ensure that the module is used in a FIPS 140-2 compliant way.

Section 2.1.1 provides algorithm-specific guidance. The requirements listed in this section are not enforced by the module and must be ensured by the module user.

Section 2.1.2 provides guidance on obtaining assurances for Digital Signature Applications.

Section 2.1.3 provides guidance on obtaining assurances for Key Agreement Applications.

Section 2.1.4 provides guidance on obtaining assurances for Key Transport Applications.

Section 2.1.5 provides information about the minimum length of passwords.

Section 2.1.6 provides general crypto user guidance.

2.1.1 Crypto User Guidance on Algorithms

The following guidance is provided for Crypto Users operating in the FIPS 140-2 approved mode.

The Crypto User must use only those algorithms approved or allowed for use in a FIPS 140-2 approved mode of operation. These algorithms are listed in:

- Table 5, Crypto-C ME FIPS 140-2-approved Algorithms
- Table 6, Crypto-C ME FIPS 140-2-allowed Algorithms.

For:

- Key Agreement:
  - For ECC based DH key agreement schemes:
    - Curves with:
      - at least 112 bits of security strength are **allowed**.
      - less than 112 bits of security strength are **not allowed**.
    - The key establishment methodology provides:
      - between 112 bits and 256 bits of encryption strength when using **approved** domain parameter size sets, as listed in Table 5.
      - between 112 and 256 bits of encryption strength when curves that are **allowed**.
      - less than 112 bits of encryption strength when using curves that are **not allowed**.
  - For FFC based DH key agreement schemes:
    - When the target security strength is greater than 112 bits, an application must use the DH FFC domain parameters from the NIST approved groups based on safe primes.
    - Generated DH FFC parameters should only be used for backwards compatibility with legacy applications.
    - When generating DH FFC domain parameters, generation shall comply with FIPS 186-4 by specifying the algorithm identifier `R_CR_ID_DH_PARAMETER_GENERATION` when creating the `R_CR` object.
    - Domain parameter size sets with:
      - L >= 2048 bits and N >= 224 bits are **allowed**
      - L < 2048 bits or N < 224 bits are **not allowed**

      Where:

      L is the bit length of the prime field size

      N is the bit length of the sub-prime field size.
    - The key establishment methodology provides:
      - 112 bits or 128 bits of encryption strength, when using **approved** domain parameter-size sets, as listed in Table 5.
      - between 112 bits and 200 bits of encryption strength when using **approved** pre-defined parameter groups, as listed in Table 5.
      - between 112 and 200 bits of encryption strength, when using **allowed** domain parameter-size sets, as listed in Table 6.
      - less than 112 bits of encryption strength when using domain parameter-size sets that are **not allowed**.

- Key Transport/Wrapping:
  - For key wrapping using AES:
    - The key establishment methodology provides between 128 and 256 bits of encryption strength.
  - For RSA Key Transport/Wrapping schemes:
    - Modulus sizes
      - greater than or equal to 2048-bits are **allowed**.
      - less than 2048-bits are **not allowed**.
    - The key establishment methodology provides:
      - 112 or 128 bits of encryption strength when using **approved** modulus sizes, as listed in Table 5.
      - between 112 and 256 bits of encryption strength when using **allowed** modulus sizes.
      - less than 112 bits of encryption strength when using modulus sizes that are **not allowed.**
- Digital Signatures.
  - An approved DRBG must be used for digital signature generation.
  - Keys used for digital signature generation and verification shall not be used for any other purpose.
  - SHA1 is **disallowed** for the generation of digital signatures.
  - For DSA:
    - When generating domain parameters, generation shall comply with FIPS 186-4 by specifying the algorithm identifier `R_CR_ID_DSA_PARAMETER_GENERATION` when creating the `R_CR` object.
    - There are no *non-approved but allowed* domain parameter set sizes. See Table 5 for approved domain parameter set sizes.
  - For ECDSA:
    - In addition to the approved named curves listed in Table 5, curves with the domain parameters generated in compliance with the rules specified in Section 6.1.1 of FIPS 186-4 are **approved** for signature verification.

      The domain parameters can be specified by name, or can be explicitly defined

      The use of these curves is also approved for signature generation if the key size is at least 224 bits.
    - There are no *non-approved but allowed* curves.
  - For RSA based schemes:
    - The length of an RSA key pair for digital signature generation must be greater than or equal to 2048 bits. For digital signature verification, the length must be greater than or equal to 2048 bits, however 1024 bits is allowed for legacy-use only. RSA keys shall have a public exponent of an odd number, equal to or greater than 65537.

- For RSASSA-PSS:
  - If the length of the RSA modulus in bits is 1024 bits, and the output length of the **approved** hash function output block is 512 bits, then the length of the salt (`sLen`) shall be `0 <= sLen <= hLen-2`

    where `hLen` is the length of the hash function output block, in bytes or octets
  - Otherwise, the length of the salt shall be `0 <= sLen <= hLen`.

- KDFs:
  - For HKDF:
    - A FIPS 140-2 approved HMAC must be used.
    - A particular key-derivation key must only be used for a single key-expansion step. For more information see SP 800-56C Rev. 1
    - The derived key must be used only as a secret key.
    - The derived key shall not be used as a key stream for a stream cipher.
    - When selecting an HMAC hash, the output block size must be equal to or greater than the desired security strength of the derived key.
  - For PBKDF2:
    - Passwords must be generated using a cryptographically secure random password generator that employs an approved DRBG.
    - The minimum password length depends on the character set chosen.

      For examples, see Information on Minimum Password Length.
    - The length of the randomly-generated portion of the salt shall be at least 16 bytes. For more information see SP 800-132.
    - The iteration count shall be selected as large as possible, a minimum of 10,000 iterations is recommended.

      See section 5.1.1.2, *Memorized Secret Verifiers*, of SP 800-63B.
    - The maximum key length is $(2^{32}-1)*b$, where b is the digest size of the hash function.
    - The key derived using PBKDF2 can be used as referred to in SP 800-132, Section 5.4, option 1 and 2.
    - Keys generated using PBKDF2 shall only be used in data storage applications.
  - For Single-step KDF:
    - A FIPS 140-2 approved hash must be used.
    - When selecting an approved hash, the output block size, in bits, must be equal to or greater than the desired security strength of the derived key.
    - The derived key must:
      - be used only as a secret key
      - not be used as a key stream for a stream cipher.

    - The maximum length of derived secret keying material is $b*(2^{32}-1)$, where b is the digest size of the hash function.
  - For SSH KDF:
    - The KDF must be used in the context of the SSH protocol.
    - A FIPS 140-2 approved hash must be used.
    - The hash function used must meet the security strength requirements of the generated keys.

    The SSH protocol has not been tested by the CAVP and CMVP.
  - For TLS 1.0, 1.1 and 1.2 KDF:
    - TLS 1.0 and 1.1 KDF is allowed only when the following conditions are satisfied:
      - The KDF is performed in the context of the TLS protocol
      - SHA-1 and HMAC are as specified in FIPS 180-4 and FIPS 198-1, respectively.
    - TLS 1.2 KDF is allowed only when the following conditions are satisfied:
      - The KDF is performed in the context of the TLS protocol
      - HMAC is as specified in FIPS 198-1
      - P_HASH uses either SHA-256, SHA-384 or SHA-512.

    For more information, see SP 800-135 Rev. 1.

    The TLS protocols have not been tested by the CAVP and CMVP.
- MAC:
  - The key length for an HMAC generation or verification must be equal to or greater than 112 bits.
  - For HMAC verification, a key length greater than or equal to 80 and less than 112 is allowed for **legacy-use**.
- Random Bit Generator:
  - Only FIPS 140-2 Approved DRBGs may be used for generation of keys, asymmetric and symmetric.
  - When using an approved DRBG, the number of bits of entropy input must be equivalent to or greater than the security strength of the keys the caller wishes to generate. For example, a 256-bit or higher entropy input when generating 256-bit AES keys.
  - When using an Approved DRBG to generate keys or FFC domain parameters, the requested security strength of the DRBG must be at least as great as the security strength of the key or domain parameters being generated. That means that an Approved DRBG with an appropriate strength must be used.

    For more information about requesting the DRBG security strength, see the **API Reference Information > Pseudo-random Number Generation** section in the *RSA BSAFE Crypto-C Micro Edition Developers Guide*.

    For further information, see **Table 3: Hash functions that can be used to provide the targeted security strengths** in SP 800-57 Part 1 Rev. 5.

- As the module does not modify the output of an Approved DRBG, any generated symmetric keys or seed values are created directly from the output of the Approved DRBG.

- Symmetric Cipher:

  - When using GCM feedback mode for symmetric encryption, the authentication tag length and authenticated data length may be specified as input parameters, but the IV must not be specified. It must be generated internally.
    IV generation operates in one of two ways:

    - In regular use the generated IV is fully random, generated by the module's approved DRBG, with a default length of 96 bits. No special considerations are required provided the system has sufficient entropy.

    - When used for TLS v1.2 protocol GCM cipher suites, as defined in RFC 5288 and allowed in section 3.3.1 of SP 800-52 Rev. 2, the four-byte salt derived from the TLS handshake process is input to the module and used to form part of the IV.

    The salt value must be assigned to the cipher structure with a call to `R_CR_set_info()` using the identifier, `R_CR_INFO_ID_CIPHER_PARTIAL_IV`, before the cipher structure is initialized. The salt is used as the first four bytes of the IV.

    The remaining eight bytes of the IV, referred to as `nonce_explicit` in RFC 5288, are generated deterministically by the module using a 64-bit global counter within the module. The module uses the current system time to initialize the counter when it is first used. The system time must be valid to prevent repetition of IVs.

    If, during a TLS connection, the `nonce_explicit` part of the IV exhausts the maximum number of possible values for a given session key, a new handshake must be performed to establish a new key.

  - AES in XTS mode is approved only for hardware storage applications.

    The two keys concatenated to create the single double-length key must be checked to ensure they are different. This is the default for the module.

    If the check is turned off by calling `R_CR_set_info()` with `R_CR_INFO_ID_CIPHER_XTS_KEY_CHECK`, AES in XTS mode is not FIPS 140-2-approved.

  - The following restrictions apply to the use of Triple-DES. For:

    - Two-key Triple-DES:

      - The use of two-key Triple-DES for encryption is **disallowed**.

      - Decryption using two-key Triple-DES is allowed for **legacy-use.**

        The user should determine the risk of accepting the decrypted information when processing more than 2^{20} blocks of data encrypted using two-key Triple-DES.

      For more information about the use of two-key Triple-DES, see SP 800-131A Rev 1.

- Three-key Triple-DES:
    - The use of three-key Triple-DES for encryption is **disallowed**.
    - Decryption using three-key Triple-DES is allowed for **legacy-use.**

      The user should determine the risk of accepting the decrypted information when processing more than 2^{20} 64-bit data blocks of data encrypted as part of one of the recognized IETF protocols. 2^{16} 64-bit data block encryptions otherwise.

  For more information about the use of three-key Triple-DES, see SP 800-67 Rev. 2.

2.1.2 Crypto User Guidance on Obtaining Assurances for Digital Signature Applications

The module provides support for the FIPS 186-4 standard for digital signatures. The following gives an overview of the assurances required by FIPS 186-4. SP 800-89 provides the methods to obtain these assurances.

The tables below describe the FIPS 186-4 requirements for signatories and verifiers and the corresponding module capabilities and recommendations.

Table 9 Signatory Requirements

FIPS 186-4 Requirement	**Module Capabilities and Recommendations**
Obtain appropriate DSA and ECDSA parameters when using DSA or ECDSA.	The generation of DSA parameters is in accordance with the FIPS 186-4 standard for the generation of probable primes. For ECDSA, use the NIST recommended curves as defined in section 2.1.1.
Obtain assurance of the validity of those parameters.	The module provides the API `R_CR_validate_key()` to validate DSA parameters for probable primes as described in FIPS 186-4. For ECDSA, use the NIST recommended curves as defined in section 2.1.1.
Obtain a digital signature key pair that is generated as specified for the appropriate digital signature algorithm.	The module generates the digital signature key pair according to the required standards. Choose a FIPS-Approved DRBG like HMAC DRBG to generate the key pair.
Obtain assurance of the validity of the public key.	The module provides the API `R_CR_validate_key()` to explicitly validate the public key according to SP 800-89.
Obtain assurance that the signatory actually possesses the associated private key.	The module verifies the signature created using the private key, but all other assurances are outside the scope of the module.

Table 10 Verifier Requirements

FIPS 186-4 Requirement	Module Capabilities and Recommendations
Obtain assurance of the signatory's claimed identity.	The module verifies the signature created using the private key, but all other assurances are outside the scope of the module.
Obtain assurance of the validity of the domain parameters for DSA and ECDSA.	The module provides the API `R_CR_validate_key()` to validate DSA parameters for probable primes as described in FIPS 186-4. For ECDSA, use the NIST recommended curves as defined in section 2.1.1.
Obtain assurance of the validity of the public key.	The module provides the API `R_CR_validate_key()` to explicitly validate the public key according to SP 800-89.
Obtain assurance that the claimed signatory actually possessed the private key that was used to generate the digital signature at the time that the signature was generated.	Outside the scope of the module.

2.1.3 Crypto User Guidance for Key Agreement Applications

The module provides support for the recommendations for key agreement in SP 800-56A Rev. 3, which provides the methods to obtain assurances of compliance.

The table below describes the SP 800-56A Rev. 3 recommendations for key agreement and the corresponding module capabilities, requirements and recommendations:

Table 11 Key Agreement Recommendations

NIST SP 800-56A Rev. 3 Recommendations	Module Capabilities, Requirements and Recommendations
Obtain domain parameters	
For schemes using FFC	FFC parameters must be selected from NIST recommended groups as defined in Section 2.1.1.
For schemes using ECC	For ECC, use the NIST recommended curves as defined in section 2.1.1.
For schemes using ECDH CDH	Both parties select approved EC parameters.
For schemes using DH	Both parties select approved FFC parameters or generate legacy parameters.
Obtain assurance of the validity of the domain parameters.	
For schemes using FFC	FFC parameters must be selected from NIST recommended groups as defined in Section 2.1.1.
For schemes using ECC	For ECC, use the NIST recommended curves as defined in section 2.1.1.

Table 11 Key Agreement Recommendations (continued)

NIST SP 800-56A Rev. 3 Recommendations	Module Capabilities, Requirements and Recommendations
Obtain a key pair from domain parameters	
For all schemes	• Both parties must use validated parameters to generate a key pair. • The module generates the key establishment key pair according to the required standards. – Choose a FIPS-Approved DRBG like HMAC DRBG to generate the key pair. • Both parties validate the key pair. – The module provides the API `R_CR_validate_key()` to explicitly validate the public and private keys according to SP 800-56A Rev. 3. – The module provides the API `R_CR_validate_key()` to explicitly validate the keypair according to the pairwise consistency requirements in SP 800-56A Rev. 3. – If the key pair is generated with an approved method, then validation is assumed.
For schemes that use static key pairs	• A public identifier must be: – authoritatively associated with the key pair. – associated with the public key to allow any peer to recognize the key pair.
For schemes that use ephemeral keys	• The key pair must be: – used only for a single agreement transaction – destroyed after use.
For schemes that generate a FFC key pair from selected parameters	• The key pair must not be used to generate a digital signature.
Receive the peer's public key	
For all schemes	The receiving party must validate the peer's public key.
For schemes that use static keys	• The receiving party must have assurance of: – the peer's ownership of the private key – the identifier is bound to the public key.
Generate the Shared Secret	
For all schemes	• The shared secret must be: – used only as input to an approved KDF – treated as a CSP and destroyed after use. • If the shared secret generation fails then the party must destroy all intermediate values.

Table 11 Key Agreement Recommendations (continued)

NIST SP 800-56A Rev. 3 Recommendations	Module Capabilities, Requirements and Recommendations
Generate and Confirm Secret Key Material	
For all schemes	• When the shared secret is used as input to the KDF the outputs must be used as secret keys • All key material must be generated before any of the keys are used • If key generation fails then the party must destroy all calculated values • The shared secret and any key material is destroyed.
For schemes that use key confirmation	• Both parties must use a common approved MAC to generate confirmation values • The MAC key will be generated as one of the key material elements • The input values for MAC tag generation must be formatted as per SP 800-56A Rev. 3 • The MAC key must be destroyed after use • If confirmation fails then destroy all calculated values – All key material is destroyed before it is used for any other purpose.

2.1.4 Crypto User Guidance on Obtaining Assurances for Key Transport Applications

The module provides support for the recommendations for key transport in SP 800-56B Rev. 2, which provides the methods to obtain these assurances.

The table below describes the SP 800-56B Rev. 2 recommendations for key transport.

Table 12 Key Transport Recommendations

NIST SP 800-56B Rev. 2 Recommendations	Module Capabilities and Recommendations
Assurance of Key-Pair Validity	The module provides the API `R_CR_validate_key()` to explicitly validate an RSA Key Pair according to SP 800-56B Rev. 2. This API performs both a pairwise consistency test and a key pair validation according to "`basic-pkv`" and "`crt_pkv`" methods.
Assurance of Public Key Validity	The module provides the API `R_CR_validate_key()` to explicitly validate the RSA public key according to SP 800-56B Rev. 2 and SP 800-89.
Assurance of Possession of Private Key	Outside the scope of the module.

2.1.5 Information on Minimum Password Length

It is assumed that generating hashes to derive keys from candidate passwords is the limiting step of brute force searching for passwords.

If an adversary has access to 1 million Graphics Processing Units (GPUs), each of which can process 1,000 million hashes per second, they can perform 6 x 10^{16} hashes per minute.

PBKDF2 Key Derivation Threat Model:

For PBKDF2, with an iteration count of 10,000, where each iteration involves an HMAC that requires at least 2 hashes, the adversary has a 1 in 100,000 chance of brute forcing a password in one minute if the password search space has 3 x 10^{17} entries.

PBKDF2 Minimum Password Length:

The minimum length (L) of a password generated using a cryptographically secure random password generator to provide a search space of S entries depends on the size (N) of the character set:

$L= \lceil \log_2 S/\log_2 N \rceil$

The following table provides examples for a password used by PBKDF2, defined in SP 800-132, where S = 3 x 10^{17}:

Character Set	N	L	Probability	
			Single Guess	One Minute
Case sensitive (a-z, A-Z)	52	11	1.33 x 10^{-19}	3.99 x 10^{-7}
Case sensitive alpha numeric	62	10	1.19 x 10^{-18}	3.57 x 10^{-6}
All ASCII printable characters except space	94	9	1.75 x 10^{-18}	5.25 x 10^{-6}

Roles Database Authentication PIN Threat Model:

The mechanism used to secure the roles databaseusesa single SHA-512 hash. The adversary has less than a 1 in 100,000 chance of brute forcing the PIN in one minute if the PIN search space has 6 x 10^{21} entries. To provide a PIN search space of S entries, each PIN must have a minimum of [$\log_2 S$] effectively random bits. For a search space greater than S = 6 x 10^{21} each PIN must have a minimum of 73 effectively random bits.

For the roles database the adversary must have less than a 1 in 1,000,000 chance of guessing the PIN in a single attempt. This can be prevented by having at least 20 random bits in the PIN.

Roles Database Minimum Password Length:

The following table provides examples for a minimum length of a randomly generated password that is to be encoded as the PIN using the corresponding character set, where S = 6 x 10^{21}:

Character Set	N	L	Probability	
			Single Guess	One Minute
Case sensitive (a-z, A-Z)	52	13	4.920 x 10^{-23}	2.952 x 10^{-6}
Case sensitive alpha numeric	62	13	4.999 x 10^{-24}	3.000 x 10^{-7}
All ASCII printable characters except space	94	12	2.101 x 10^{-24}	1.261 x 10^{-7}

2.1.6 General Crypto User Guidance

Crypto-C ME users should take care to zeroize CSPs when they are no longer needed. For more information on clearing sensitive data, see section 1.6.5 and the relevant API documentation in the *RSA BSAFE Crypto-C Micro Edition Developer Guide*.

2.2 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 in which a user can operate:

Table 13 Services Authorized for Roles

Role	Authorized Services

Crypto Officer
`R_FIPS140_ROLE_OFFICER`
	All services including
	`R_PROV_FIPS140_self_tests_full()`
	`R_PROV_FIPS140_init_roles()`
	`R_PROV_FIPS140_set_roles_file()`
	`R_PROV_FIPS140_set_roles_file_w()`
	`R_PROV_FIPS140_set_pin()`
	`R_PROV_FIPS140_set_pin_with_token()`

Crypto User `R_FIPS140_ROLE_USER`	All services **except** those listed for the Officer role, above.

The complete list of the functionality available is outlined in Services.

2.3 Modes of Operation

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

Table 14 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 5. 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_PROV_FIPS140_self_tests_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.

2.4 Operating the Module

Crypto-C ME operates in a FIPS 140-2-approved mode on startup, providing access to only the algorithms listed in Table 5 from the FIPS 140-2 provider set against the library context. To restrict the module to an alternative set of algorithms, call `R_LIB_CTX_set_mode()` with one of the mode filters listed in Table 14.

To disable FIPS 140-2 mode, call `R_LIB_CTX_set_mode()` with `NULL` to put Crypto-C ME into an unrestricted mode that provides access to all cryptographic algorithms available from the FIPS 140-2 provider.

To retrieve the current Crypto-C ME FIPS 140-2 mode, call `R_LIB_CTX_get_mode()`.

To run self-tests on the FIPS 140-2 module, the application must ensure that there are no cryptographic operations using the module.

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

The user of Crypto-C ME links with the `ccme` static library for their platform. At run time, the static library loads the `cryptocme` master shared library, which then loads all of the resource shared libraries. For more information, see **Get Stated with Crypto-C ME > Binary Installation > Installed Library Files** in the *RSA BSAFE Crypto-C Micro Edition Developers Guide*.

The current Crypto-C ME role is determined by calling `R_PROV_FIPS140_get_info()` with `R_FIPS_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 13.

2.5 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.

2.5.1 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 Requirements for Security” in RFC 4086 and SP 800-90A Rev. 1.

3 Services

The following is the list of authenticated and unauthenticated services provided by Crypto-C ME. Authenticated services can only be used by users authenticated in the Crypto Officer or Crypto User role. Unauthenticated services can be used by any user.

For more information about authentication of roles, see Roles, Services and Authentication. For more information about individual functions, see the *RSA BSAFE Crypto-C Micro Edition Developers Guide*.

3.1 Authenticated Services

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_delete
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_delete
R_CR_derive_key
R_CR_derive_key_data
R_CR_digest
R_CR_digest_final
R_CR_digest_init
R_CR_digest_size
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_asn1_params
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_kdf_extract
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_init_PKEY
R_CR_keywrap_unwrap_init_SKEY

```
R_CR_keywrap_unwrap_PKEY
R_CR_keywrap_unwrap_SKEY
R_CR_keywrap_wrap
R_CR_keywrap_wrap_init
R_CR_keywrap_wrap_init_PKEY
R_CR_keywrap_wrap_init_SKEY
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_new_from_R_ALG_PARAMS
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_asn1_params
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_get_desc_string
R_CR_validate_get_string
R_CR_validate_init_PKEY
R_CR_validate_key
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
R_PKEY_cmp
R_PKEY_copy
R_PKEY_CTX_add_filter
R_PKEY_CTX_delete
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_PKEY_CTX
R_PKEY_get_type
R_PKEY_identify
R_PKEY_is_matching_public_key
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_to_bio
R_PKEY_public_to_file
R_PKEY_reference_inc
R_PKEY_remove
R_PKEY_SEARCH_add_filter
R_PKEY_SEARCH_delete
R_PKEY_SEARCH_free
R_PKEY_SEARCH_init
R_PKEY_SEARCH_new
R_PKEY_SEARCH_next
R_PKEY_set_info
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_to_string
R_PROV_FIPS140_assume_role
R_PROV_FIPS140_authenticate_role
R_PROV_FIPS140_authenticate_role_with_token
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_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_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_remove
R_SKEY_SEARCH_add_filter
R_SKEY_SEARCH_delete
R_SKEY_SEARCH_free
R_SKEY_SEARCH_init
R_SKEY_SEARCH_new
R_SKEY_SEARCH_next
R_SKEY_set_info
R_SKEY_store

3.2 Unauthenticated Services

R_ALG_PARAMS_asym_from_binary
R_ALG_PARAMS_cipher_from_binary
R_ALG_PARAMS_ctrl
R_ALG_PARAMS_delete
R_ALG_PARAMS_digest_from_binary
R_ALG_PARAMS_free
R_ALG_PARAMS_from_binary
R_ALG_PARAMS_get_binary
R_ALG_PARAMS_get_info
R_ALG_PARAMS_kdf_from_binary
R_ALG_PARAMS_keywrap_from_binary
R_ALG_PARAMS_new
R_ALG_PARAMS_new_from_R_CR
R_ALG_PARAMS_peek_error
R_ALG_PARAMS_peek_error_string
R_ALG_PARAMS_pop_error
R_ALG_PARAMS_pop_error_string
R_ALG_PARAMS_ref_inc
R_ALG_PARAMS_set_info
R_ALG_PARAMS_signature_from_binary
R_ALG_PARAMS_signature_get_info
R_ALG_PARAMS_to_binary

R_ALG_signature_info
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_BIO_can_read
R_BIO_can_write
R_BIO_clear_flags
R_BIO_clear_retry_flags
R_BIO_ctrl
R_BIO_delete
R_BIO_dump
R_BIO_dump_format
R_BIO_f_buffer_new
R_BIO_f_callback_new
R_BIO_f_cipher_new
R_BIO_f_der_new
R_BIO_f_digest_new
R_BIO_find_type
R_BIO_flags_to_string
R_BIO_f_pem_new
R_BIO_f_prefix_new
R_BIO_free
R_BIO_free_all
R_BIO_f_rwmerge_new
R_BIO_get_flags
R_BIO_get_retry_flags
R_BIO_get_retry_reason
R_BIO_gets
R_BIO_mem
R_BIO_method_name
R_BIO_method_type
R_BIO_new_fd
R_BIO_new_fd_ef
R_BIO_new_fd_file
R_BIO_new_fd_file_ef
R_BIO_new_file
R_BIO_new_file_ef
R_BIO_new_file_w
R_BIO_new_file_w_ef
R_BIO_new_fp
R_BIO_new_fp_ef
R_BIO_new_mem
R_BIO_new_mem_ef
R_BIO_pem_finish_section
R_BIO_pem_next_section
R_BIO_pem_start_section
R_BIO_pop
R_BIO_pop_delete
R_BIO_printf
R_BIO_print_hex
R_BIO_puts
R_BIO_read
R_BIO_reference
R_BIO_reference_inc
R_BIO_retry_type
R_BIO_select
R_BIO_set_flags
R_BIO_set_retry_read
R_BIO_set_retry_small_buffer
R_BIO_set_retry_special
R_BIO_set_retry_write
R_BIO_s_fd_new
R_BIO_s_fd_open
R_BIO_s_fd_open_w
R_BIO_s_file_new
R_BIO_s_file_open
R_BIO_s_file_open_w
R_BIO_s_fmem_new
R_BIO_should_io_special
R_BIO_should_read
R_BIO_should_retry
R_BIO_should_small_buffer
R_BIO_should_write
R_BIO_s_mem_new
R_BIO_s_null_new
R_BIO_wait_readable
R_BIO_wait_writeable
R_BIO_write
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_FILTER_sort

R_FORMAT_from_string
R_FORMAT_to_string
R_GBL_ERR_STATE_add_error_data
R_LIB_CTX_add_filter
R_LIB_CTX_add_provider
R_LIB_CTX_add_resource
R_LIB_CTX_add_resources
R_LIB_CTX_delete
R_LIB_CTX_dup
R_LIB_CTX_dup_ef
R_LIB_CTX_free
R_LIB_CTX_get_info
R_LIB_CTX_get_mode
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_library_info
R_library_info_type_from_string
R_library_info_type_to_string
R_library_version
R_LOCK_add
R_LOCK_delete
R_LOCK_exec
R_LOCK_free
R_LOCK_lock
R_LOCK_new
R_LOCK_unlock
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_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_PROV_ctrl
R_PROV_delete
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_from_label
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_manufacturer_id
R_PROV_PKCS11_get_token_model
R_PROV_PKCS11_get_token_serial_number
R_PROV_PKCS11_has_token_login_pin
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_delete
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_STACK_cat
R_STACK_clear
R_STACK_clear_arg
R_STACK_delete
R_STACK_delete_all
R_STACK_delete_all_arg
R_STACK_delete_ptr
R_STACK_dup
R_STACK_dup_ef
R_STACK_find
R_STACK_for_each
R_STACK_free
R_STACK_insert
R_STACK_lfind
R_STACK_move
R_STACK_new
R_STACK_new_ef
R_STACK_pop
R_STACK_pop_free
R_STACK_pop_free_arg
R_STACK_push
R_STACK_set
R_STACK_set_cmp_func
R_STACK_shift
R_STACK_unshift
R_STACK_zero
R_STATE_cleanup
R_STATE_disable_cpu_features
R_STATE_init
R_STATE_init_defaults
R_STATE_init_defaults_mt
R_STATE_refresh
R_STATE_set_fork_check
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
R_THREAD_create
R_THREAD_id
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_delete
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_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
```

4 Acronyms and Definitions

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

Table 15 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. 4.1.5
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, middle person attack and timing 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.
CAVP	Cryptographic Algorithm Validation Program (CAVP) provides validation testing of FIPS-approved and NIST-recommended cryptographic algorithms and their individual components.
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.
CDH	The cofactor ECC Diffie-Hellman key-agreement primitive defined in the SP 800-56A series.
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.
CSP	A Critical Security Parameters is security related information, such as keys or passwords, whose disclosure or modification can compromise security.
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.

Table 15 Acronyms and Definitions (continued)

Term	Definition
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 with eight parity bits. See also Triple-DES.
DESX	A variant of the DES symmetric key algorithm intended to increase the complexity of a brute force attack.
Diffie-Hellman	The Diffie-Hellman (DH) 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.
ECB	Electronic Codebook. A mode of encryption, which divides a message into blocks and encrypts each block separately.
ECC	Elliptic Curve Cryptography (ECC): the public-key cryptographic methods using operations in an elliptic curve group. ECC keys are used in several algorithms including ECDSA, ECDH and ECDHC. An individual ECC key must not be used for multiple purpose, for example, signing and key agreement.
ECDH	Elliptic Curve Diffie-Hellman key agreement algorithm. This algorithm uses a key-agreement primitive that does not employ the elliptic curve’s cofactor.
ECDHC	Elliptic Curve Diffie-Hellman with Cofactor key agreement algorithm. This algorithm employs the CDH primitive.
ECDSA	Elliptic Curve Digital Signature Algorithm.
ECIES	Elliptic Curve Integrated Encryption Scheme.
Encryption	The transformation of plaintext into an apparently less readable form (called ciphertext) through a mathematical process. The ciphertext can be read by anyone who has the key and decrypts (undoes the encryption) the ciphertext.
FFC	Finite Field Cryptography (FFC): the public-key cryptographic methods using operations in a multiplicative group of a finite field. FFC keys are use in algorithms including DSA and Diffie-Hellman.
FIPS	Federal Information Processing Standards.

Table 15 Acronyms and Definitions (continued)

Term	Definition
FIPS 180-4	Federal Information Processing Standards Publication: Secure Hash Standard (SHS).
FIPS 186-2	Federal Information Processing Standards Publication:
FIPS 186-4	Federal Information Processing Standards Publication: Digital Signature Standard (DSS).
FIPS 198-1	Federal Information Processing Standards Publication: The Keyed-Hash Message Authentication Code (HMAC).
FIPS 202	Federal Information Processing Standards Publication: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.
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.
HKDF	HMAC-based Extract-and Expand KDF. HKDF is a two-step key derivation function, where the first step, extraction, transforms a shared secret into a key-derivation key. The second step, expansion, uses the key-derivation key to derive an output key
HMAC	Keyed-Hashing for Message Authentication Code.
HMAC DRBG	HMAC Deterministic Random Bit Generator.
IG	Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program.
IV	Initialization Vector. Used as a seed value for an encryption or MAC operation.
JCMVP	Japan Cryptographic Module Validation Program.
KAT	Known Answer Test.
Key	A string of bits used by cryptographic algorithms. There are a variety of cryptographic key types. These keys might be used for operations such as encryption or decryption, cryptographic signing or verification, or key agreement. Some types of keys are intended to be kept secret, and other keys are intended to be public.
Key wrapping	A method of encrypting key data for protection on untrusted storage devices or during transmission over an insecure channel.

Table 15 Acronyms and Definitions (continued)

Term	Definition
L	The bit length of the prime field size.
MAC	Message Authentication Code.
MD2	A message digest algorithm, which hashes an arbitrary-length input into a 16-byte digest.
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.
N	The bit length of the subprime field size.
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.
P_HASH	A function that uses the HMAC-HASH as the core function in its construction. Specified in RFC 2246 and RFC 5246.
PBKDF1	Password-based Key Derivation Function 1. A method of password-based key derivation defined in RFC 2988, 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, originally defined in RFC 2988, which applies a Message Authentication Code (MAC) algorithm to derive the key. In RFC 2988 the PRF used by PBKDF2 is specified as SHA-1. SP 800-132 approves PBKDF2 where the PRF may be any FIPS approved hash function. In this document PBKDF2 represents the expanded specification provided in SP 800-132.
PRF	PseudoRandom Function
Private Key	The secret key in public key cryptography. Primarily used for decryption but also used for generation of digital signatures.
PRNG	Pseudo-random Number Generator.
Public Key	The public key in public key cryptography. Primarily used for encryption but also verification of digital signatures.

Table 15 Acronyms and Definitions (continued)

Term	Definition
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.
RFC 2246	The TLS Protocol Version 1.0.
RFC 2313	PKCS #1: RSA Encryption.
RFC 2998	PKCS #5: Password-Based Cryptography Specification.
RFC 4086	Randomness Requirements for Security.
RFC 4346	The Transport Layer Security (TLS) Protocol Version 1.1.
RFC 5246	The Transport Layer Security (TLS) Protocol Version 1.2.
RFC 5488	AES Galois Counter Mode (GCM) Cipher Suites for TLS.
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.
SHA-3	SHA-3 is a family of hash algorithms which include SHA-3-224, SHA-3-256, SHA-3-384 and SHA-3-512 bits. It is an alternative to SHA-2, as no significant attacks on SHA-2 are currently known.
SEED	A symmetric key algorithm developed by the Korean Information Security Agency.

Table 15 Acronyms and Definitions (continued)

Term	Definition
SP 800-38A	NIST Special Publication 800-38A: Recommendation for Block 2001 Edition Cipher Modes of Operation Methods and Techniques.
SP 800-38C	NIST Special Publication 800-38C: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality.
SP 800-38D	NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.
SP 800-38E	NIST Special Publication 800-38E: Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices.
SP 800-38F	NIST Special Publication 800-38F: Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping.
SP 800-56A Rev. 3	NIST Special Publication 800-56A Revision 3: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography.
SP 800-56B	NIST Special Publication 800-56B Revision 2: Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography.
SP 800-56C	NIST Special Publication 800-56C Revision 1: Recommendation for Key-Derivation Methods in Key-Establishment Schemes.
SP 800-57 Part 1 Rev. 5	NIST Special Publication 800-57 Part 1 Revision 5: Recommendation for Key Management.
SP 800-67 Rev. 2	NIST Special Publication 800-67 revision 2: Recommendations for The Triple Data Encryption Block Cipher.
SP 800-89	NIST Special Publication 800-89: Recommendation for Obtaining Assurances for Digital Signature Applications.
SP 800-90A Rev. 1	NIST Special Publication 800-90A Revision 1: Recommendation for Random Number Generation Using Deterministic Random Bit Generators.
SP 800-108	NIST Special Publication 800-108: Recommendation for Key Derivation Using Pseudorandom Functions (Revised).
SP 800-131A	NIST Special Publication 800-131A Revision 2: Transitioning the Use of Cryptographic Algorithms and Key Lengths.
SP 800-132	NIST Special Publication 800-132: Recommendation for Password-Based Key Derivation.
SP 800-133 Rev. 2	NIST Special Publication 800-133 Revision 2: Recommendation for Cryptographic Key Generation.
SP 800-135 Rev. 1	NIST Special Publication 800-135 Revision 1: Recommendation for Existing Application-Specific Key Derivation Functions.
Triple-DES	A variant of DES. A symmetric encryption algorithm which uses three 56-bit keys with eight parity bits each.

Table 15 Acronyms and Definitions (continued)

Term	Definition
XTS	XEX-based Tweaked Codebook mode with ciphertext stealing. A mode of encryption used with AES.