FIPS 140-2 Non-Proprietary Security Policy

CryptoComply for NSS

Software Version 5.0

Document Version 1.0

December 1, 2021

SafeLogic Inc.
530 Lytton Ave., Suite 200
Palo Alto, CA 94301
www.safelogic.com

Overview

This document provides a non-proprietary FIPS 140-2 Security Policy for CryptoComply for NSS.

SafeLogic's CryptoComply for NSS is designed to provide FIPS 140-2 validated cryptographic functionality and is available for licensing. For more information, visit www.safelogic.com/software.

Table of Contents

Overview **2**

1 Introduction **6**

1.1 About FIPS 140 ..... *6*

1.2 About this Document *6*

1.3 External Resources *6*

1.4 Notices *6*

2 CryptoComply for NSS **7**

2.1 Cryptographic Module Specification *7*

2.1.1 Validation Level Detail 8

2.1.2 Modes of Operation 8

2.1.3 Approved Cryptographic Algorithms 9

2.1.4 Non-Approved but Allowed Cryptographic Algorithms 12

2.1.5 Non-Approved Mode of Operation 12

2.2 Critical Security Parameters and Public Keys *14*

2.2.1 Critical Security Parameters 14

2.2.2 Random Number Generation 16

2.2.3 Key/CSP Storage 16

2.2.4 Key/CSP Zeroization 16

2.3 Module Interfaces *18*

2.3.1 Inhibition of Data Output 19

2.3.2 Disconnecting the Output Data Path from the Key Process 20

2.4 Roles, Services, and Authentication *21*

2.4.1 Roles 21

2.4.2 Assumption of Roles 21

2.4.3 Strength of Authentication Mechanism 22

2.4.4 Services 23

2.5 Physical Security *31*

2.6 Operational Environment *31*

2.7 Self-Tests *32*

2.7.1 Power-Up Self-Tests 32

2.7.2 Conditional Self-Tests 33

2.8 Mitigation of Other Attacks *34*

3 Security Rules and Guidance **36**

3.1 Crypto Officer Guidance *36*

3.1.1 Access to Audit Data 37

3.2 User Guidance *37*

3.2.1 TLS Operations 38

3.2.2 RSA and DSA Keys 39

3.2.3 Triple-DES keys 39

3.3 Handling Self-Test Errors *39*

3.4 Basic Enforcement *39*

4 References and Acronyms **40**

4.1 References 40
4.2 Acronyms 41

List of Tables

Table 1 - Validation Level by FIPS 140-2 Section 8
Table 2 - FIPS Approved Algorithm Certificates 9
Table 3 - Non-Approved but Allowed Cryptographic Algorithms 12
Table 4 - Non-Approved Cryptographic Functions for Use in Non-Approved mode Only 12
Table 5 - Critical Security Parameters 14
Table 6 - Logical Interface/Physical Interface Mapping 19
Table 7 - Description of Roles 21
Table 8 - Services in Approved Mode 24
Table 9 - Services in Non-Approved Mode 30
Table 10 - FIPS Tested Configurations 32
Table 11 - Power-Up Self-Tests 32
Table 12 - Conditional Self-Tests 34
Table 13 - Mitigation of Other Attacks 34
Table 14 - References 40
Table 15 - Acronyms and Terms 41

List of Figures

Figure 1 - Module Boundary and Interfaces Diagram 18

1 Introduction

1.1 About FIPS 140

Federal Information Processing Standards Publication 140-2 — Security Requirements for Cryptographic Modules specifies requirements for cryptographic modules to be deployed in a Sensitive but Unclassified environment. The National Institute of Standards and Technology (NIST) and Canadian Centre for Cyber Security (CCCS) Cryptographic Module Validation Program (CMVP) run the FIPS 140 program. The National Voluntary Laboratory Accreditation Program (NVLAP) accredits independent testing labs to perform FIPS 140 testing; the CMVP validates modules meeting FIPS 140 validation. *Validated* is the term given to a module that is documented and tested against the FIPS 140 criteria.

More information is available on the CMVP website at https://csrc.nist.gov/projects/cryptographic-module-validation-program.

1.2 About this Document

This non-proprietary Cryptographic Module Security Policy for CryptoComply for NSS from SafeLogic Inc. ("SafeLogic") provides an overview of the product and a high-level description of how it meets the overall Level 1 security requirements of FIPS 140-2.

CryptoComply for NSS may also be referred to as the "module" in this document.

1.3 External Resources

The SafeLogic website (www.safelogic.com) contains information on SafeLogic services and products. The Cryptographic Module Validation Program website contains links to the FIPS 140-2 certificate and SafeLogic contact information.

1.4 Notices

2 CryptoComply for NSS

2.1 Cryptographic Module Specification

CryptoComply for NSS is a software library designed to provide FIPS 140-2 validated cryptographic functionality and is available for licensing.

The module's software version is 5.0. The module is an open-source, general-purpose cryptographic library, with an API based on the industry standard PKCS #11 version 2.20. It combines a vertical stack of Linux components intended to limit the external interface each separate component may provide.

The module is a software module that relies on the physical characteristics of the host platform. The module's physical boundary is defined by the enclosure of the host platform, which is the General Purpose Device that the module is installed on. For the purposes of FIPS 140-2 validation, the module's embodiment type is defined as multi-chip standalone.

All operations of the module occur via calls from host applications and their respective internal daemons/processes. As such there are no untrusted services calling the services of the module.

The module's logical cryptographic boundary is the shared library files and their integrity check signature files as listed below:

- `libsoftokn3.so`
- `libsoftokn3.chk`
- `libnssdbm3.so`
- `libnssdbm3.chk`
- `libfreeblpriv3.so`
- `libfreeblpriv3.chk`

2.1.1 Validation Level Detail

The following table lists the module's level of validation for each area in FIPS 140-2:

Table 1 - Validation Level by FIPS 140-2 Section

FIPS 140-2 Section Title	Validation Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	2
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
Electromagnetic Interference/Electromagnetic Compatibility	1
Self-Tests	1
Design Assurance	3
Mitigation of Other Attacks	1

2.1.2 Modes of Operation

The module supports two modes of operation: FIPS Approved mode and non-Approved mode. The module will be in FIPS Approved mode when all power up self-tests have completed successfully, and only Approved or allowed algorithms are invoked. See Table 2 below for a list of the supported Approved algorithms and Table 3 for allowed algorithms. The non-Approved mode is entered when a non-Approved algorithm is invoked. See Table 4 for a list of non-Approved algorithms.

2.1.3 Approved Cryptographic Algorithms

The module's cryptographic algorithm implementations have received the following certificate numbers from the Cryptographic Algorithm Validation Program (CAVP).

Table 2 - FIPS Approved Algorithm Certificates

CAVP Cert.	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
C786 C794	**AES**	FIPS 197 SP 800-38A SP 800-38F	CBC, CTR (ext only), ECB, KW[1] tested without and with AES-NI	128, 192, 256	Encryption, Decryption
Vendor affirmed	**CKG**	SP 800-133	Symmetric keys and asymmetric seeds are the unmodified output of the SP 800-90A DRBG (ref. SP section 2.2.2)		Key Generation
C786	**CVL: KAS-ECC Component**	SP 800-56A	Key Pair Generation, Partial PKV: Full unified scheme (Initiator/Responder) with parameter sets EC (P-256, SHA-256), ED (P-384, SHA-384), EC (P-521, SHA-512)		Used by allowed EC Diffie-Hellman
C786	**CVL: KAS-FFC Component**	SP 800-56A	Key Pair Generation: dhEphem scheme (Initiator/Responder) with parameter sets FB (SHA-224), FC (SHA-256)		Used by allowed Diffie-Hellman
C786	**CVL: KDF, Application-Specific[2]**	SP 800-135	TLS v1.0/1.1 KDF TLS v1.2 KDF	SHA-256, SHA-384, SHA-512	TLS pre-master secret and master secret

[1] The AES key wrapping provides between 128 and 256 bits of encryption strength.
[2] These protocols have not been reviewed or tested by the CAVP and CMVP.

CAVP Cert.	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
C786	**DRBG**	SP 800-90A	Hash_DRBG (SHA-256)	256 bits entropy input	Random Bit Generation The module generates cryptographic keys whose strengths are modified by available entropy.
C786	**DSA**	FIPS 186-4	Key Generation: (2048, 224), (2048, 256), (3072, 256) Signature Generation: (2048, 224) (SHA-224, SHA-256, SHA-384, SHA-512) (2048, 256) (SHA-256, SHA-384, SHA-512) (3072, 256) (SHA-256, SHA-384, SHA-512) PQG Verification, Signature Verification: (1024, 160) (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) (2048, 224) (SHA-224, SHA-256, SHA-384, SHA-512) (2048, 256) (SHA-256, SHA-384, SHA-512) (3072, 256) (SHA-256, SHA-384, SHA-512)		Digital Signature Services
C786	**ECDSA**	FIPS 186-4	Key Pair Generation, Public Key Validation P-256, P-384, P-521 (Secret Generation Mode: Extra Bits) Signature Generation: P-256, P-384, P-521 (SHA-224, SHA-256, SHA-384, SHA-512) Signature Verification: P-256, P-384, P-521 (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512)		Digital Signature Services

CAVP Cert.	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
C786	**HMAC**	FIPS 198-1	HMAC-SHA-1 HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512	Key Size Ranges Tested: KS<BS, KS=BS, KS>BS	HMAC Generation, Authentication
C786 C794	**KTS**[3]	SP 800-38F	AES KW (without or with AES-NI)	128, 192, 256	Key establishment methodology provides between 128 and 256 bits of encryption strength
C786	**RSA**	FIPS 186-4 PKCS #1 v1.5	Key Generation: 2048, 3072 bits (primality per Table C.3) Signature Generation (PKCS #1 v1.5): 2048, 3072, 4096[4] bits (SHA-224, SHA-256, SHA-384, SHA-512) Signature Verification (PKCS #1 v1.5): 1024, 2048, 3072 bits (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512)		Digital Signature Services
C786	**SHS**	FIPS 180-4	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512	Byte-only	Digital Signature Services, non-Digital Signature Applications
C786	**Triple-DES**	SP 800-67	TECB, TCBC and CTR (ext only)	3-key[5]	Encryption, Decryption

[3] For use with system-level key establishment; not used by the module to establish keys within the module
[4] Tested under 186-2, per IG G.18 (including additional comment 1)
[5] Each 3-key Triple-DES key shall not be used to encrypt more than 2^{16} 64-bit blocks of data

2.1.4 Non-Approved but Allowed Cryptographic Algorithms

The module supports the following FIPS 140-2 non-Approved but allowed algorithms that may be used in the FIPS Approved mode of operation.

Table 3 - Non-Approved but Allowed Cryptographic Algorithms

Algorithm	Use
Diffie-Hellman Key Agreement (≥ 2048)	Using KAS-FFC Component (CVL Cert. #C786). Key agreement; key establishment methodology provides between 112 and 256 bits of encryption strength. Non-compliant if less than 112 bits of encryption strength. For use with system-level key establishment; not used by the module to establish keys within the module.
Elliptic Curve Diffie-Hellman (P-256, P-384 and P-521)	Using KAS-ECC Component (CVL Cert. #C786). Key agreement; key establishment methodology provides between 128 and 256 bits of encryption strength For use with system-level key establishment; not used by the module to establish keys within the module.
MD5 within TLS	Message digest used in TLS only [ref. IG 1.23 example 2a]
NDRNG	Used for seeding NIST SP 800-90A DRBG
RSA Key Wrapping (≥ 2048)	Key wrapping; key establishment methodology provides between 112 and 256 bits of encryption strength. Non-compliant if less than 112 bits of encryption strength.

2.1.5 Non-Approved Mode of Operation

The module supports a non-Approved mode of operation. The algorithms listed in this section are not to be used by the operator in the FIPS Approved mode of operation.

Table 4 - Non-Approved Cryptographic Functions for Use in Non-Approved mode Only

Algorithm	Use
AES CTS	Encryption, Decryption
AES-GCM (non-compliant)	Encryption, Decryption (non-compliant with IG A.5, CAVS tested with Cert. # C786)
Camellia	Encryption, Decryption
DES	Encryption, Decryption
Diffie-Hellman (non-compliant[6])	Key Agreement
DSA (non-compliant[7])	Public Key Cryptography

[6] Support for additional key sizes (key agreement with key sizes less than 2048 bits)
[7] Support for additional key sizes (key generation/signature generation with key sizes less than 2048 bits, signature verification with key sizes less than 1024 bits)

Algorithm	Use
J-PAKE	Key Agreement
Key wrapping	Non-SP 800-38F AES and Triple-DES key wrapping
MD2	Hashing
MD5	Hashing
RC2	Encryption, Decryption
RC4	Encryption, Decryption
RC5	Encryption, Decryption
RSA (non-compliant[8])	Public Key Cryptography
SEED	Encryption, Decryption
Triple-DES with 2-key	Encryption, Decryption, key wrapping using two-key Triple-DES

[8] Support for additional key sizes (key generation/signature generation/key wrapping with key sizes less than 2048 bits, signature verification with key sizes less than 1024 bits)

2.2 Critical Security Parameters and Public Keys

2.2.1 Critical Security Parameters

The table below provides a complete list of Critical Security Parameters (CSPs) used within the module.

Table 5 - Critical Security Parameters

CSP	Generation	Storage	Entry/Output	Zeroization
AES keys (128, 192, 256 bits)	Use of NIST SP 800-90A DRBG	Application memory or key database	Encrypted through key wrapping using FC_WrapKey	Automatically zeroized when freeing the cipher handle
Triple-DES keys (192 bits)	Use of NIST SP 800-90A DRBG	Application memory or key database	Encrypted through key wrapping using FC_WrapKey	Automatically zeroized when freeing the cipher handle
DSA private keys (2048, 3072 bits)	186-4 (use of NIST SP 800-90A DRBG)	Application memory or key database	Encrypted through key wrapping using FC_WrapKey	Automatically zeroized when freeing the cipher handle
ECDSA private keys (P-256, P-384, P-521)	186-4 (use of NIST SP 800-90A DRBG)	Application memory or key database	Encrypted through key wrapping using FC_WrapKey	Automatically zeroized when freeing the cipher handle
RSA private keys (2048, 3072 bits)	186-4 (use of NIST SP 800-90A DRBG)	Application memory or key database	Encrypted through key wrapping using FC_WrapKey	Automatically zeroized when freeing the cipher handle
HMAC keys (≥ 112 bits)	Use of NIST SP 800-90A DRBG	Application memory or key database	Encrypted through key wrapping using FC_WrapKey	Automatically zeroized when freeing the cipher handle
DRBG entropy input string and seed	Obtained from NDRNG	Application memory	N/A	Automatically zeroized when freeing the DRBG handle
DRBG V and C values	Derived from the entropy input string as defined in NIST SP 800-90A	Application memory	N/A	Automatically zeroized when freeing the DRBG handle
TLS pre-master secret	Use of NIST SP 800-90A DRBG in Diffie-Hellman or EC Diffie-Hellman key agreement scheme	Application memory	N/A	Automatically zeroized when freeing the cipher handle
TLS master secret	Derived from TLS pre-master secret by using key derivation (TLS KDF)	Application memory	N/A	Automatically zeroized when freeing the cipher handle

CSP	Generation	Storage	Entry/Output	Zeroization
Diffie-Hellman private components (2048 - 15360 bits)	Use of NIST SP 800-90A DRBG in Diffie-Hellman key agreement scheme	Application memory	N/A	Automatically zeroized when freeing the cipher handle
EC Diffie-Hellman private components (P-256, P-384, P-521)	Use of NIST SP 800-90A DRBG in EC Diffie-Hellman key agreement scheme	Application memory	N/A	Automatically zeroized when freeing the cipher handle
User Passwords	N/A (supplied by the calling application)	Application memory or key database in salted form	N/A (input through API parameter)	Automatically zeroized when the module is re-initialized or overwritten when the user changes its password

2.2.2 Random Number Generation

The module employs a NIST SP 800-90A Hash_DRBG with SHA-256 as a random number generator.

For symmetric keys, the generated key is an unmodified output from the SP 800-90A DRBG. For generating asymmetric keys, the module implements key generation services compliant with FIPS 186-4, and the seed (i.e. the random value) used in asymmetric key generation is an unmodified output from the SP 800-90A DRBG. In accordance with FIPS 140-2 IG D.12, the cryptographic module performs Cryptographic Key Generation (CKG) as per SP 800-133 (vendor affirmed).

The module uses NDRNG from /dev/urandom as a source of entropy for seeding the DRBG. The NDRNG is provided by the operating environment (i.e., Linux RNG), which is within the module's physical boundary but outside of its logical boundary. The NDRNG provides at least 130 bits of entropy to the DRBG.

CAVEAT: The module generates cryptographic keys whose strengths are modified by available entropy

Reseeding is performed by pulling more data from /dev/urandom. A product using the module should periodically reseed the module's random number generator with unpredictable noise by calling FC_SeedRandom. After 2^{48} calls to the random number generator the module reseeds automatically.

The module performs DRBG health testing as specified in section 11.3 of NIST SP 800-90A and the continuous random number generator test (CRNGT) on the output of DRBG to ensure that consecutive random numbers do not repeat. The underlying operating system performs the continuous test on the NDRNG. If CRNGT fails, the operating system kernel panics and the module is not available for use.

2.2.3 Key/CSP Storage

The module employs the cryptographic keys and CSPs in the FIPS Approved mode operation as listed in Table 5 - Critical Security Parameters. The module does not perform persistent storage for any keys or CSPs.

Note that the private key database (provided with the files key3.db/key4.db) is within the module's physical boundary but outside its logical boundary.

2.2.4 Key/CSP Zeroization

The application that uses the module is responsible for appropriate zeroization of the key material. The module provides zeroization methods to clear the memory region previously occupied by a plaintext secret key, private key, or password. A plaintext secret or private key is zeroized when it is passed to a FC_DestroyObject call. All plaintext secret and private keys must be zeroized when the module is shut down (with a FC_Finalize call), reinitialized (with a FC_InitToken call), or when the session is closed (with a FC_Close Session or FC_CloseAllSessions call). All zeroization is to be performed by storing the value 0 into every byte of memory region that is previously occupied by a plaintext secret key, private key, or

password. Zeroization is performed in a time that is not sufficient to compromise plaintext secret keys, private keys, or passwords.

2.3 Module Interfaces

The figure below shows the module's physical and logical block diagram:

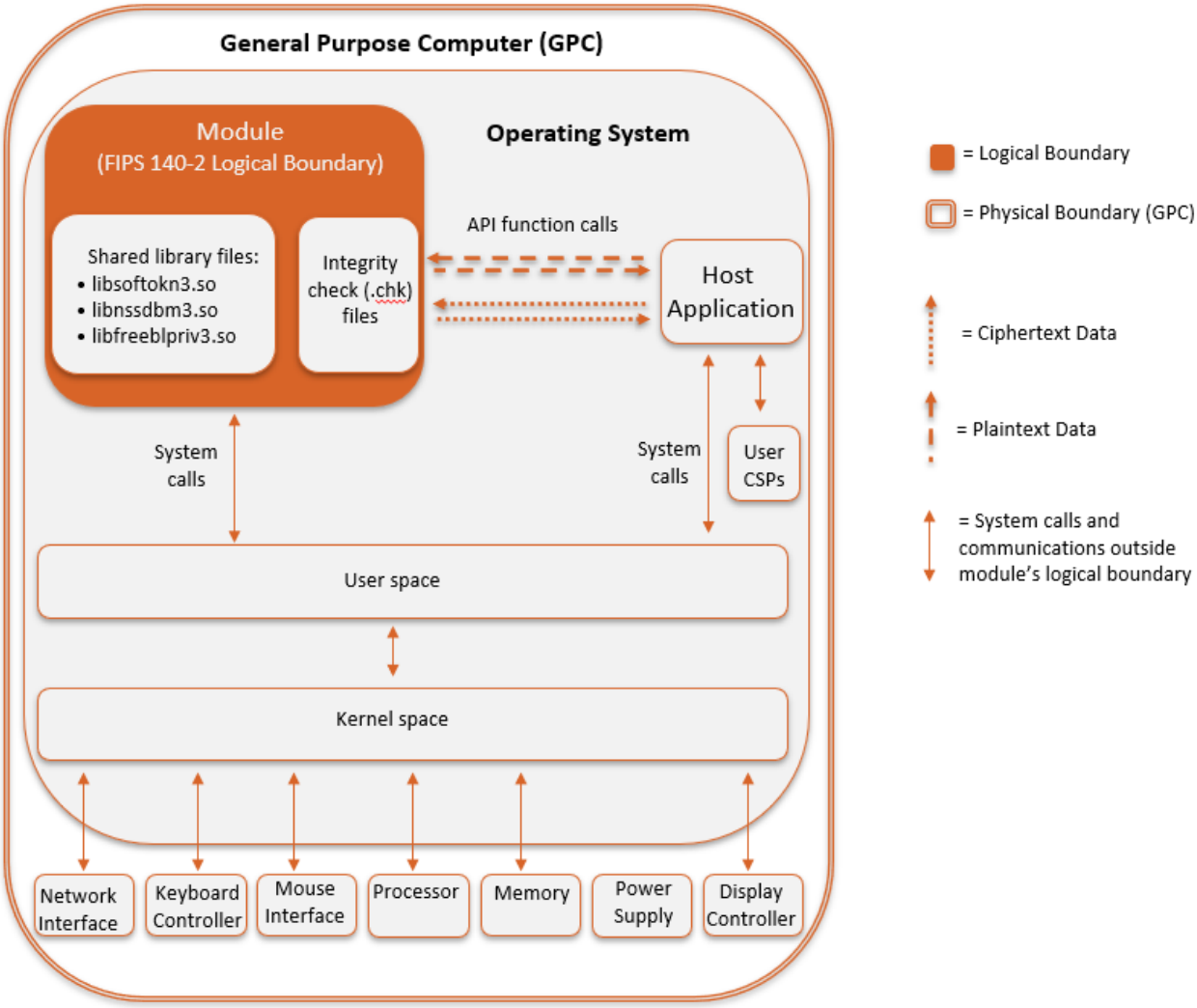

Figure 1 - Module Boundary and Interfaces Diagram

The module's physical boundary is the boundary of the General Purpose Computer (GPC) that the module is installed on, which includes a processor and memory. The interfaces (ports) for the physical boundary include the computer's network port, keyboard port, mouse port, power plug and display. When operational, the module does not transmit any information across these physical ports because it is a software cryptographic module. Therefore, the module's interfaces are purely logical.

The logical interface is a C-language Application Programming Interface (API) following the PKCS #11 specification (Cryptoki), the database files in a kernel file system, and the environment variables. The API itself defines the module's logical boundary, i.e. all access to the module is through this API. The API provides functions that may be called by an application (see Section 2.4.4 – Services for the list of

available functions). The module distinguishes between logical interfaces by logically separating the information according to the defined API.

The API provided by the module is mapped onto the FIPS 140-2 logical interfaces, which relate to the module's callable interface as follows.

Table 6 - Logical Interface/Physical Interface Mapping

FIPS 140-2 Interface	Module Logical Interface	GPC Physical Interface
Data Input	API input parameters – plaintext and/or ciphertext data - and database files in kernel file system	Network Interface
Data Output	API output parameters and return values – plaintext and/or ciphertext data – and database files in kernel file system	Network Interface
Control Input	API method calls – method calls, or input parameters, that specify commands and/or control data used to control the operation of the module (/proc/sys/crypto/fips_enabled)	Network Interface, Keyboard Interface, Mouse Interface
Status Output	API output parameters and return/error codes that provide status information used to indicate the state of the module	Display Controller, Network Interface
Power	None	Power Supply

The module uses different function arguments for input and output to distinguish between data input, control input, data output, and status output, to disconnect the logical paths followed by data/control input entering the module and data/status output exiting the module. The output data path is provided by the data interfaces and is logically disconnected from processes performing key generation or zeroization. No key information will be output through the data output interface when the module zeroizes keys.

The module does not use the same buffer for input and output. After the module is done with an input buffer that holds security-related information, it always zeroizes the buffer so that if the memory is later reused as an output buffer, no sensitive information can be inadvertently leaked.

2.3.1 Inhibition of Data Output

All data output via the data output interface is inhibited when the module is performing self-tests or in the Error state.

During self-tests: All data output via the data output interface is inhibited while self-tests are executed.

In Error state: The Boolean state variable sftk fatalError tracks whether the module is in the Error state. Most PKCS #11 functions, including all the functions that output data via the data output interface, check the sftk_fatalError state variable and, if it is true, return the CKR_DEVICE_ERROR error code immediately. Only the functions that shut down and restart the module, reinitialize the module, or output status information can be invoked in the Error state. These functions are FC_GetFunctionList,

FC_Initialize, FC_Finalize, FC_GetInfo, FC_GetSlotList, FC_GetSlotinfo, FC_GetTokeninfo, FC_InitToken, FC_CloseSession, FC_CloseAllSessions, and FC_WaitForSlotEvent.

2.3.2 Disconnecting the Output Data Path from the Key Process

During key generation and key zeroization, the module may perform audit logging, but the audit records do not contain sensitive information. The module does not return the function output arguments until the key generation or key zeroization is finished. Therefore, the logical paths used by output data exiting the module are logically disconnected from the processes/threads performing key generation and key zeroization.

2.4 Roles, Services, and Authentication

2.4.1 Roles

The module supports two distinct operator roles, which are the User and Crypto Officer (CO). The cryptographic module implicitly maps the two roles to the services. A user is considered the owner of the thread that instantiates the module and, therefore, only one concurrent user is allowed.

The module does not support a Maintenance role or bypass capability.

Table 7 - Description of Roles

Role	Role Description	Authentication Type
Crypto Officer (CO)	The CO installs and initializes the module. The CO can access other general-purpose services (such as message digest and random number generation services) and status services of the module. The CO does not have access to any service that utilizes the secret or private keys of the module. The CO must control the access to the module both before and after installation, including the management of physical access to the computer, executing the module code, as well as management of the security facilities provided by the operating system.	Role-Based Authentication
User	The User role has access to all cryptographically secure services which use the secret or private keys of the module. It is also responsible for the retrieval, updating, and deletion of keys from the private key database.	Role-Based Authentication

2.4.2 Assumption of Roles

2.4.2.1 Assumption of CO Role

The CO role is implicitly assumed by an operator while installing the module by following the instructions in Section 3.1 and while performing other CO services on the module.

2.4.2.2 Assumption of User Role

The module implements a password-based authentication for the User role. To perform any security services under the User role, an operator must log into the module and complete an authentication procedure using the password information unique to the User role operator.

The password is passed to the module via the API function as an input argument and will not be displayed. The return value of the function is the only feedback mechanism, which does not provide any

information that could be used to guess or determine the User's password. The password is initialized by the CO role as part of the module initialization and can be changed by the User role operator.

If a User role service is called before the operator is authenticated, the module returns the CKR_USER_NOT_LOGGED_IN error code. The operator must call the FC_Login function to provide the required authentication.

Once a password has been established for the module, the user is allowed to use the security services if and only if the user is successfully authenticated to the module. Password establishment and authentication are required for the operation of the module. When module is powered off, the result of previous authentication will be cleared and the user needs to be re-authenticated.

2.4.3 Strength of Authentication Mechanism

The module imposes the following requirements on the password. These requirements are enforced by the module on password initialization or change.

- The password must be at least seven characters long.
- The password must consist of characters from three or more character classes. We define five character classes below. If an ASCII uppercase letter is the first character of the password, the uppercase letter is not counted toward its character class. Similarly, if a digit is the last character of the password, the digit is not counted toward its character class.
    - digits (0-9)
    - ASCII lowercase letters (a-z)
    - ASCII uppercase (A-Z)
    - ASCII non-alphanumeric characters (space and other ASCII special characters such as '$' and '!')
    - non-ASCII characters (Latin characters such as 'é', 'ß'; Greek characters such as 'Ω', 'θ', other non-ASCII special characters such as '¿').

To estimate the maximum probability that a random guess of the password will succeed, we assume that:

- The characters of the password are independent from each other.
- The password contains the smallest combination of the character classes, which is five digits, one ASCII lowercase letter and one ASCII uppercase letter. The probability to guess every character successfully is $(1/10)^5*(1/26)*(1/26) = 1/67,600,000$.

Since the password can contain seven characters from any three or more of the aforementioned five character classes, the probability that a random guess of the password will succeed is less than or equal to 1/67,600,000. This probability of success is smaller than the FIPS 140-2 required threshold of 1/1,000,000.

After each failed authentication attempt, the module inserts a one-second delay before returning to the caller, allowing at most 60 authentication attempts during a one-minute period. Therefore, the probability of a successful random guess of the password during a one-minute period is less than or equal to 60*1/67,600,000 = 0.089* (1/100,000), which is smaller than the FIPS 140-2 required threshold of 1/100,000.

2.4.4 Services

The module has a set of API functions denoted by FC_xxx. All the API functions are listed in Table 8 - Services.

Among the module's API functions, only FC_GetFunctionlist is exported and therefore callable by its name. All the other API functions must be called via the function pointers returned by FC_GetFunctionlist. It returns a CK_FUNCTION_LIST structure containing function pointers named C_xxx such as C_Initialize and C_Finalize. The C_xxx function pointers in the CK_FUNCTION_LIST structure returned by FC_GetFunctionlist point to the FC_xxx functions .

The following convention is used to describe API function calls. Here FC_Initialize is used as an example:

- When "call FC_Initialize" is mentioned, the technical equivalent of "call the FC_Initialize function via the C_Initialize function pointer in the CK_FUNCTION_LIST structure returned by FC_GetFunctionlist" is implied.

The module supports Crypto-Officer services which require no operator authentication, and User services which require operator authentication. Crypto-Officer services do not require access to the secret and private keys and other CSPs associated with the User. The message digesting services are available to Crypto-Officer only when CSPs are not accessed[9]. User services which access CSPs (e.g., FC_GenerateKey, FC_GenerateKeyPair) require operator authentication.

[9] The message digesting functions (except FC_DigestKey) that do not use any keys of the module can be accessed by the Crypto-Officer role and do not require authentication to the module. The FC_DigestKey API function computes the message digest (hash) of the value of a secret key, so it is available only to the User role.

2.4.4.1 FIPS Approved Mode Services

All services implemented by the module in FIPS Approved mode are listed in Table 8 - Services. The table includes a description of each service, service availability to the Crypto Officer and User, and CSP access. Modes of CSP access shown in the table are defined as:

- **G** = Generate: The module generates the CSP.
- **R** = Read: The module reads the CSP. The read access is typically performed before the module uses the CSP.
- **E** = Execute: The module executes using the CSP.
- **W** = Write: The module writes the CSP. The write access is typically performed after a CSP is imported into the module, when the module generates a CSP, or when the module overwrites an existing CSP.
- **Z** = Zeroize: The module zeroizes the CSP.

Table 8 - Services in Approved Mode

Service	Function	Description	CO	User	Key, CSP Access
Get Function List	• FC_GetFunctionList	• Return a pointer to the list of function pointers for the operational mode	X		• N/A
Module Initialization	• FC_InitToken • FC_InitPIN	• Initialize or re-initialize a token • Initialize the user's password, i.e., set the user's initial password	X		• Z: User password and all keys • W: user password
General Purpose	• FC_Initialize • FC_Finalize • FC_GetInfo	• Initialize the module library • Finalize (shut down) the module library • Obtain general information about the module library	X		• N/A • Z: all keys • N/A

Service	Function	Description	CO	User	Key, CSP Access
Slot and Token Management	• FC_GetSLotList • FC_GetSlotInfo • FC_GetTokenInfo • FC_GetMechanismList • FC_GetMechanismInfo	• Obtain a list of slots in the system • Obtain information about a particular slot • Obtain information about the token (This function provides the Show Status service) • Obtain a list of mechanisms (cryptographic algorithms) supported by a token • Obtain information about a particular mechanism	X		• N/A • N/A • N/A • N/A • N/A
	• FC_SetPIN	• Change the user's password		X	• R, W: User password
Session Management	• FC_OpenSession • FC_CloseSession • FC_CloseAllSessions • FC_GetSessionInfo • FC_GetOperationState • FC_SetOperationState	• Open a connection (session) between an application and a particular token • Close a session • Close all sessions with a token • Obtain information about the session (This function provides the Show Status service) • Save the state of the cryptographic operations in a session (This function is only implemented for message digest operations) • Restore the state of the cryptographic operations in a session (This function is only implemented for message digest operations)	X		• N/A • Z: all keys for the session • Z: all keys • N/A • N/A • N/A
	• FC_Login • FC_Logout	• Log into a token • Log out from a token		X	• R, W, E: User password • N/A

Service	Function	Description	CO	User	Key, CSP Access
Object Management	• FC_CreateObject • FC_CopyObject • FC_DestroyObject • FC_GetObjectSize • FC_GetAttributeValue • FC_SetAttributeValue • FC_FindObjectsInit • FC_FindObjects • FC_FindObjectsFinal	• Create a new object • Create a copy of an object • Destroy an object • Obtain the size of an object in bytes • Obtain an attribute value of an object • Modify an attribute value of an object • Initialize an object search operation • Continue an object search operation • Finish an object search operation		X	• W: key[10] • R, W: original key, new key • Z: key • R: key • R: key • W: key • N/A • R: keys matching the search criteria • N/A
Encryption and Decryption	• FC_EncryptInit • FC_Encrypt • FC_EncryptUpdate • FC_EncryptFinal • FC_DecryptInit • FC_Decrypt • FC_DecryptUpdate • FC_DecryptFinal	• Initialize an encryption operation • Encrypt single-part data • Continue a multiple-part encryption operation • Finish a multiple-part encryption operation • Initialize a decryption operation • Decrypt single-part encrypted data • Continue a multiple-part decryption operation • Finish a multiple-part decryption operation		X	• R: AES/TDES key • R: AES/TDES key • R: AES/TDES key • R: AES/TDES key • R: AES/TDES key • R: AES/TDES key • R: AES/TDES key • R: AES/TDES key • R: AES/TDES key
Message Digest	• FC_DigestInit • FC_Digest • FC_DigestUpdate • FC_DigestFinal	• Initialize a message-digesting operation • Digest single-part data • Continue a multiple-part digesting operation • Finish a multiple-part digesting operation	X		• N/A • N/A • N/A • N/A

[10] "key" may represent a secret keys or public/private key pair, with CSPs as specified in Table 5

Service	Function	Description	CO	User	Key, CSP Access
	• FC_DigestKey	• Continue a multiple-part message-digestion operation by digesting the value of a secret key as part of the data already digested		X	• R: HMAC key
Signature Generation and Verification	• FC_SignInit • FC_Sign • FC_SignUpdate • FC_SignFinal • FC_SignRecoverInit • FC_SignRecover • FC_VerifyInit • FC_Verify • FC_VerifyUpdate • FC_VerifyFinal • FC_VerifyRecoverInit • FC_VerifyRecover	• Initialize a signature operation • Sign single-part data • Continue a multiple-part signature operation • Finish a multiple-part signature operation • Initialize a signature operation, where the data can be recovered from the signature • Sign single-part data, where the data can be recovered from the signature • Initialize a verification operation • Verify a signature on single-part data • Continue a multiple-part verification operation • Finish a multiple-part verification operation • Initialize a verification operation, where the data is recovered from the signature • Verify a signature on single-part data, where the data is recovered from the signature		X	• R: DSA/ECSDA/RSA private key, HMAC key • R: DSA/ECSDA/RSA private key, HMAC key • R: DSA/ECSDA/RSA private key, HMAC key • R: DSA/ECSDA/RSA private key, HMAC key • R: DSA/ECSDA/RSA private key • R: DSA/ECSDA/RSA private key • R: DSA/ECSDA/RSA private key, HMAC key • R: DSA/ECSDA/RSA private key, HMAC key • R: DSA/ECSDA/RSA private key, HMAC key • R: DSA/ECSDA/RSA private key, HMAC key • R: DSA/ECSDA/RSA private key • R: DSA/ECSDA/RSA private key

Service	Function	Description	CO	User	Key, CSP Access
Dual-function Cryptographic Operations	• FC_DigestEncryptUPdate • FC_DecryptDigestUpdate • FC_SignEncryptUpdate • FC_DecryptVerifyUpdate	• Continue a multiple-part digesting and encryption operation • Continue a multiple-part decryption and digesting operation • Continue a multiple-part signing and encryption operation • Continue a multiple-part decryption and verification operation		X	• R: AES/TDES key • R: AES/TDES key • R: DSA/ECSDA/RSA private key, HMAC/AES/TDES key • R: DSA/ECSDA/RSA private key, HMAC/AES/TDES key
Key Management	• FC_GenerateKey • FC_GenerateKeyPair • FC_WrapKey • FC_UnwrapKey • FC_DeriveKey	• Generate a secret key (Also used by TLS to generate a pre-master secret) • Generate a public/private key pair (This function performs the pair-wise consistency tests) • Wrap (encrypt) a key per either: (1) SP 800-38F (AES) (2) RSA encryption • Unwrap (decrypt) a key per either: (1) SP 800-38F (AES) (2) RSA decryption • Derive a key from TLS master secret which is derived from TLS pre-master secret		X	• W: HMAC/AES/TDES key, TLS pre-master secret • W: DSA/ECDSA/RSA key pair, Diffie-Hellman/EC Diffie-Hellman key pair • R: key used for wrapping, wrapped key • R: key used for unwrapping, wrapped key • R: TLS pre-master secret, R/W: TLS master secret, W: derived key
Random Number Generation	• FC_SeedRandom • FC_GenerateRandom	• Mix in additional seed material to the random number generator • Generate random data (This function performs the continuous random generator test)	X		• R/W: entropy input string, seed, DRBG V and C values • R/W Random data, DRBG V and C values
Parallel Function Management	• FC_GetFunctionStatus • FC_CancelFunction	• A legacy function, which simply returns the value 0x00000051 (function not parallel) • A legacy function, which simply returns the value 0x00000051 (function not parallel)	X		• N/A • N/A

Service	Function	Description	CO	User	Key, CSP Access
Self-Tests	• N/A	• The self-tests are performed automatically when loading the module	X		• R: DSA public key for integrity test
Zeroization	• FC_InitToken • FC_Finalize • FC_CloseSession • FC_CloseAllSessions	All CSPs are automatically zeroized when freeing the cipher handle	X		Z: all secret or private keys and User password
	• FC_DestroyObject	• All CSPs are automatically zeroized when freeing the cipher handle		X	• Z: all secret or private keys and User password

2.4.4.2 Non-Approved Mode Services

Table 9 lists all the services available in non-Approved mode with API function and the non-Approved algorithm(s) that the function may invoke. Please note that the functions are the same as the ones listed in Table 8, but the non-Approved algorithms specified below are invoked. Please also refer to Table 4 for the non-Approved algorithms. If any service invokes the non-Approved algorithms, then the module will enter non-Approved mode implicitly.

Table 9 - Services in Non-Approved Mode

Service	Function	Non-Approved Algorithm Invoked
Encryption and Decryption	• FC_EncryptInit • FC_Encrypt • FC_EncryptUpdate • FC_EncryptFinal • FC_DecryptInit • FC_Decrypt • FC_DecryptUpdate • FC_DecryptFinal	AES GCM mode, AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
Message Digest	• FC_DigestInit • FC_Digest • FC_DigestUpdate • FC_DigestFinal • FC_DigestKey	MD2, MD5
Signature Generation and Verification	• FC_SignInit • FC_Sign • FC_SignUpdate • FC_SignFinal • FC_SignRecoverInit • FC_SignRecover	DSA or RSA signature generation with non-compliant key size < 2048
	• FC_VerifyInit • FC_Verify • FC_VerificyUpdate • FC_VerifyFinal • FC_VerifyRecoverInit • FC_VerifyRecover	DSA or RSA signature verification with non-compliant key size < 1024
Dual-function Cryptographic Operations	• FC_DigestEncryptUpdate	MD2, MD5, AES GCM mode, AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
	• FC_DecryptDigestUpdate	MD2, MD5, AES GCM mode, AES CTS mode, Camellia, DES RC2, RC4, RC5, SEED

Service	Function	Non-Approved Algorithm Invoked
	• FC_SignEncryptUpdate	DSA or RSA signature generation with non-compliant key size < 2048, AES GCM mode, AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
	• FC_DecryptVerifyUpdate	DSA or RSA signature verification with non-compliant key size < 1024, AES GCM mode, AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
Key Management	• FC_GenerateKeyPair	DSA or RSA key pair generation with non-compliant key size < 2048
	• FC_KeyWrapKey	Triple-DES key wrapping (encrypt) using two-key or three-key Triple-DES, RSA key wrapping (encrypt) with non-compliant key size < 2048, non-SP 800-38F AES key wrapping (encrypt)
	• FC_UnwrapKey	Triple-DES key wrapping (decrypt) using two-key or three-key Triple-DES, RSA key wrapping (decrypt) with non-compliant key size, non-SP 800-38F AES key unwrapping (dencrypt)
	• FC_DeriveKey	Diffie-Hellman key agreement with non-compliant key size < 2048, J-PAKE key agreement

2.5 Physical Security

The module is a software-only module and does not have physical security mechanisms.

2.6 Operational Environment

The module operates in a modifiable operational environment under the FIPS 140-2 definitions.

The module runs on a GPC running one of the operating systems specified in this section, i.e. the approved operational environments. Each approved operating system manages processes and threads in a logically separated manner. The module's user is considered the owner of the calling application (thread) that instantiates the module, therefore only one concurrent user is allowed.

In FIPS Approved mode, the ptrace system call, the debugger gdb, and the strace shall not be used. In addition, other tracing mechanisms offered by the Linux environment, such as ftrace or systemtap, shall not be used.

The module was tested on the following platforms:

Table 10 - FIPS Tested Configurations

Operating System	Hardware	Processor	PAA (AES-NI)
Oracle Linux 7.6 64-bit	Oracle Server X7-2	AMD® EPYC® 7551	Yes
Oracle Linux 7.6 64-bit	Oracle Server X7-2	AMD® EPYC® 7551	No
Oracle Linux 7.6 64-bit	Oracle Server X7-2	Intel® Xeon® Silver 4114	Yes
Oracle Linux 7.6 64-bit	Oracle Server X7-2	Intel® Xeon® Silver 4114	No

FIPS 140-2 validation compliance is maintained for other compatible operating systems (in single user mode) where the module source code is unmodified, and the requirements outlined in NIST IG G.5 are met. No claim can be made as to the correct operation of the module or the security strengths of the generated keys when ported to an operational environment that is not listed on the validation certificate.

The GPC(s) used during testing met Federal Communications Commission (FCC) FCC Electromagnetic Interference (EMI) and Electromagnetic Compatibility (EMC) requirements for business use as defined by 47 Code of Federal Regulations, Part15, Subpart B.

2.7 Self-Tests

Each time the module is powered up, it tests that the cryptographic algorithms still operate correctly, and that sensitive data has not been damaged. Power-up self-tests are available on demand by reinitializing (power cycling) the module.

On power-up or reset (initialization or reinitialization), the module automatically performs the self-tests that are described in Table 11 - Power-Up Self-Tests without requiring any operator intervention. During the power-up self-tests, no cryptographic operations are available, and all input or output is inhibited. All power-up self-tests must be completed successfully before the module enters operational mode and cryptographic operations become available. If any of the power-up self-test fails, the module enters the Error state. The module returns the error code CKR_DEVICE_ERROR to the calling application to indicate the Error state. The module needs to be reinitialized to recover from Error state.

2.7.1 Power-Up Self-Tests

Table 11 - Power-Up Self-Tests

Test Target	Test Details
Module Integrity	• DSA signature verification (2048, SHA-256) (Cert. #C786)

Test Target	Test Details
AES	• AES ECB encrypt KAT • AES ECB decrypt KAT • AES CBC encrypt KAT • AES CBC decrypt KAT
DRBG	• Hash_DRBG with SHA-256 KAT
DSA	• DSA sign KAT • DSA verify KAT
ECDSA	• ECDSA sign KAT • ECDSA verify KAT
HMAC	• HMAC-SHA-1 KAT • HMAC-SHA-224 KAT • HMAC-SHA-256 KAT • HMAC-SHA-384 KAT • HMAC-SHA-512 KAT
RSA	• RSA encrypt KAT • RSA decrypt KAT • RSA sign KAT • RSA verify KAT
SHS	• SHA-1 KAT • SHA-224 KAT • SHA-256 KAT • SHA-384 KAT • SHA-512 KAT
Triple-DES	• Triple-DES ECB encrypt KAT • Triple-DES ECB decrypt KAT • Triple-DES CBC encrypt KAT • Triple-DES CBC decrypt KAT

2.7.2 Conditional Self-Tests

The following table provides the lists of Pairwise Consistency Tests (PCT) and Continuous Random Number Generator Test (CRNGT) as the conditional self-tests. If any of the conditional test fails, the module enters the Error state. It returns the error code CKR_DEVICE_ERROR to the calling application to indicate the Error state. The module needs to be reinitialized to recover from the Error state.

Table 12 - Conditional Self-Tests

Test Target	Test Details
DRBG	CRNGT[11]
DRBG	Health testing as specified in Section 11.3 of NIST SP 800-90A[12]
DSA	PCT for DSA key generation
ECDSA	PCT for ECDSA key generation
RSA	PCT for RSA key generation

2.8 Mitigation of Other Attacks

The module is designed to mitigate the following attacks:

Table 13 - Mitigation of Other Attacks

Attack	Mitigation Mechanism	Specific Limit
Timing attacks on RSA	**RSA Blinding** Timing attack on RSA was first demonstrated by Paul Kocher in 1996 [P. Kocher], who contributed the mitigation code to our module. Most recently Boneh and Brumley [D. Boneh] showed that RSA blinding is an effective defense against timing attacks on RSA.	None
Cache-timing attacks on the modular exponentiation operation used In RSA and DSA	**Cache invariant module exponentiation** This is a variant of a modular exponentiation implementation that Colin Percival [C. Percival] showed to defend against cache-timing attacks	This mechanism requires intimate knowledge of the cache line sizes of the processor. The mechanism may be ineffective when the module is running on a processor whose cache line sizes are unknown.

[11] The underlying operating system performs the continuous test on the NDRNG. If CRNGT fails, the operating system kernel panics and the module is not available for use. Refer to Section 2.2.2.
[12] These tests are performed both at start-up and conditionally at run-time.

Attack	Mitigation Mechanism	Specific Limit
Arithmetic errors in RSA signatures	**Double-checking RSA signatures** Arithmetic errors in RSA signatures might leak the private key. Ferguson and Schneier [N. Ferguson] recommend that every RSA signature generation should verify the signature just generated.	None

3 Security Rules and Guidance

3.1 Crypto Officer Guidance

The module is provided directly to solution developers and is not available for direct download to the general public. Only the compiled module is provided to solution developers. The module shall be installed on an operating system specified in Section 2.6 or one where portability is maintained.

The version of the RPMs containing the FIPS validated module is stated in Section 2.1. The RPM packages forming the module can be installed by standard tools recommended for the installation of RPM packages in Enterprise Linux systems (for example, yum, rpm, and the RHN remote management tool). The CO shall verify the hash of the RPM package to confirm a proper download.

In addition, to support the module, the NSPR library must be installed that is offered by the underlying operating system.

Only the cipher types listed in section 2.1.3 and 2.1.4 are allowed to be used in FIPS approved mode.

To bring the module into FIPS Approved mode, perform the following steps:

1. Install the dracut-fips package:
   ```
   # yum install dracut-fips
   ```
2. Recreate the INITRAMFS image:
   ```
   # dracut -f
   ```

After regenerating the initramfs, the Crypto Officer has to append the following string to the kernel command line by changing the setting in the boot loader:

```
fips=1
```

If /boot or /boot/efi resides on a separate partition, the kernel parameter boot=<partition of /boot or /boot/efi> must be supplied. The partition can be identified with the respective commands

```
"df /boot"
```

Or

```
"df /boot/efi"
```

For example:

```
$ df /boot

Filesystem  1k-blocks   Used        Available   Use%        Mounted on
/dev/sda1   233191      30454       190296      14%         /boot
```

The partition of /boot is located on /dev/sda1 in this example. Therefore, the following string needs to be appended to the kernel command line:

```
"boot=/dev/sda1"
```

Reboot to apply these settings.

If an application that uses the module for its cryptography is put into a chroot environment, the Crypto Officer must ensure one of the above methods is available to the module from within the chroot environment to ensure entry in FIPS Approved mode. Failure to do so will not allow the application to properly enter FIPS Approved mode.

3.1.1 Access to Audit Data

The module may use the Unix syslog function and the audit mechanism provided by the operating system to audit events. Auditing is turned off by default. Auditing capability must be turned on as part of the initialization procedures by setting the environment variable NSS_ENABLE_AUDIT to 1. The Crypto-Officer must also configure the operating system's audit mechanism.

The module uses the syslog function to audit events, so the audit data are stored in the system log. Only the root user can modify the system log. On some platforms, only the root user can read the system log; on other platforms, all users can read the system log. The system log is usually under /var/log directory. The exact location of the system log is specified in the /ext/syslog.conf file. The module uses the default user facility and the info, warning, and err severity levels for its log messages.

The module can also be configured to use the audit mechanism provided by the operating system to audit events. The audit data would then be stored in the system audit log. Only the root user can read or modify the system audit log. To turn on this capability it is necessary to create a symbolic link from the library file /usr/lib/libaudit.so.0 to /usr/lib.libaudit.so.1.0.0 (on 32-bit platforms) and /usr/lib64/libaudit.so.0 to /usr/lib64/libaudit.so.1.0.0 (on 64-bit platforms).

3.2 User Guidance

The module must be operated in FIPS Approved mode to ensure that FIPS 140-2 validated cryptographic algorithms and security functions are used. To run the module in FIPS Approved mode, only the FIPS Approved services listed in Table 8 with the Approved or allowed cryptographic algorithms/security functions listed in Table 2 and Table 3 shall be used.

The following module initialization steps must be followed by the Crypto-Officer before starting to use the module:

- Set the environment variables NSS_ENABLE_AUDIT to 1 before using the module with an application
- Use the application to get the function pointer list using the API "FC_GetFunctionList".

- Use the API FC_Initialize to initialize the module and ensure that it returns CKR_OK. A return code other than CKR_OK means the module is not initialized correctly, and in that case, the module must be reset and initialized again.
- For the first login, provide a NULL password and login using the function pointer C_Login, which will in-turn call FC_Login API of the module. This is required to set the module's initial NSS User password.
- Now, set the module's initial NSS User role password using the function pointer C_INITPIN. This will call the module's API FC_InitPIN API. Then, logout using the function pointer C_Logout, which will call the module's API FC_Logout.
- The module's NSS User role can now be assumed on the module by logging in using the User password. The Crypto-Officer role can be implicitly assumed by performing the Crypto-Officer services as listed in section 2.4.4.

The module can be configured to use different private key database formats: key3.db or key4.db. "key3.db" format is based on the Berkeley DataBase engine and should not be used by more than one process concurrently. "key4.db" format is based on SQL DataBase engine and can be used concurrently by multiple processes. Both databases are considered outside the module's logical boundary and all data stored in these databases is considered stored in plaintext. The interface code of the module that accesses data stored in the database is considered part of the cryptographic boundary.

Secret and private keys, plaintext passwords and other security-relevant data items are maintained under the control of the cryptographic module. Secret and private keys must be passed to the calling application in encrypted (wrapped) form with FC_WrapKey and entered from the calling application in encrypted form with FC_UnwrapKey. The key transport methods allowed for this purpose in FIPS Approved mode are SP 800-38F based AES key wrapping and RSA key wrapping using the corresponding Approved modes and key sizes. Note: If the secret and private keys passed to the calling application are encrypted using a symmetric key algorithm, the encryption key may be derived from a password. In such a case, they should be considered to be in plaintext form in the FIPS Approved mode.

Automated key transport methods must use FC_WrapKey and FC_UnwrapKey to output or input secret and private keys from or to the module. Note these are available in non-Approved mode only.

All cryptographic keys used in the FIPS Approved mode of operation must be generated in the FIPS Approved mode or imported while running in the FIPS Approved mode.

3.2.1 TLS Operations

The module does not implement the TLS protocol. The module implements the cryptographic operations, including TLS-specific key generation and derivation operations, which can be used to implement the TLS protocol.

3.2.2 RSA and DSA Keys

The module allows the use of 1024-bit RSA and DSA keys for legacy purposes including signature generation, which is disallowed for use in FIPS Approved mode as per NIST SP 800-131A. Therefore, the cryptographic operations with the non-Approved key sizes will result in the module operating in non-Approved mode implicitly.

3.2.3 Triple-DES keys

According to IG A.13, the same Triple-DES key shall not be used to encrypt more than 2^{16} 64-bit blocks of data. Encrypting greater than 216 blocks will result in the module operating in non-Approved mode implicitly. It is the the User's responsible for ensuring the module's compliance with this requirement

3.3 Handling Self-Test Errors

When the module enters the Error state, it needs to be reinitialized to resume normal operation. Reinitialization is accomplished by calling FC_Finalize followed by FC_Initialize.

3.4 Basic Enforcement

The module design corresponds to the module security rules. This section documents the security rules enforced by the cryptographic module to implement the security requirements of this FIPS 140-2 Level 1 module.

1. The module provides two distinct operator roles: User and Cryptographic Officer.
2. The operator may command the module to perform the power up self-tests by cycling power or resetting the module.
3. Power-up self-tests do not require any operator action.
4. Data output is inhibited during key generation, self-tests, zeroization, and error states.
5. Status information does not contain CSPs or sensitive data that if misused could lead to a compromise of the module.
6. There are no restrictions on which keys or CSPs are zeroized by the zeroization service.
7. The module does not support concurrent operators.
8. The module does not have any external input/output devices used for entry/output of data.
9. The module does not enter or output plaintext CSPs from the module's physical boundary.
10. The module does not output intermediate key values.

4 References and Acronyms

4.1 References

Table 14 - References

Abbreviation	Full Specification Name
FIPS 140-2	*Security Requirements for Cryptographic modules, May 25, 2001*
FIPS 180-4	*Secure Hash Standard (SHS)*
FIPS 186-4	*Digital Signature Standard (DSS)*
FIPS 197	*Advanced Encryption Standard*
FIPS 198-1	*The Keyed-Hash Message Authentication Code (HMAC)*
IG	*Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program*
PKCS#1 v2.1	*RSA Cryptography Standard*
PKCS#5	*Password-Based Cryptography Standard*
PKCS#11	*"PKCS #11 v2.20: Cryptographic Token Interface Standard", 2004.*
SP 800-38A	*Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode*
SP 800-38D	*Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*
SP 800-38F	*Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*
SP 800-56A	*Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*
SP 800-67	*Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*
SP 800-90A	*Recommendation for Random Number Generation Using Deterministic Random Bit Generators*
PKCS 11 v2.20	*RSA Laboratories, "PKCS #11 v2.20: Cryptographic Token Interface Standard", 2004.*
P.Kocher	*P.Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", CRYPTO '96 Lecture Notes in Computer Science, Vol. 1109, pp. 104-113, Springer Verlag, 1996.* http://www.cryptography.com/timingattack/
D. Boneh	*D. Boneh and D. Brumley. "Remote Timing Attacks are Practical",* http://crypto.stanford.edu/~dabo/abstracts/ssl-timing.html
C. Percival	*C. Percival, "Cache Missing for Fun and Profit",* http://www.daemonology.net/papers/htt.pdf
N. Ferguson	*N. Ferguson and B. Schneier, Practical Cryptography, Sec. 16.1.4 "Checking RSA Signatures", p. 286, Wiley Publishing, Inc., 2003.*

4.2 Acronyms

The following table defines acronyms found in this document:

Table 15 - Acronyms and Terms

Acronym	Term
AES	Advanced Encryption Standard
AES-NI	Intel Advanced Encryption Standard New Instructions
API	Application Programming Interface
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CCCS	Canadian Centre for Cyber Security
CCM	Counter with CBC-MAC
CMVP	Cryptographic Module Validation Program
CO	Crypto Officer
CSP	Critical Security Parameter
CTR	Counter Mode
CVL	Component Validation List
DES	Data Encryption Standard
DRAM	Dynamic Random Access Memory
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FCC	Federal Communications Commission
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
GMAC	Galois Message Authentication Code
GPC	General Purpose Computer
HMAC	(Keyed-) Hash Message Authentication Code
IG	Implementation Guidance
IV	Initialization Vector
KAS	Key Agreement Scheme
KAT	Known Answer Test
KDF	Key Derivation Function
MAC	Message Authentication Code
MD5	Message Digest Algorithm MD5

Acronym	Term
N/A	Not Applicable
NDRNG	Non Deterministic Random Number Generator
NIST	National Institute of Standards and Technology
NSS	Network Security Services
OFB	Output Feedback
OS	Operating System
PKCS	Public-Key Cryptography Standards
RSA	Rivest, Shamir, and Adleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
TCBC	TDEA Cipher-Block Chaining
TDEA	Triple Data Encryption Algorithm
TDES	Triple Data Encryption Standard
TECB	TDEA Electronic Codebook
TOFB	TDEA Output Feedback
TLS	Transport Layer Security
USB	Universal Serial Bus