ORACLE
Linux

FIPS 140-2 Non-Proprietary Security Policy

Oracle Linux 7 NSS Cryptographic Module

FIPS 140-2 Level 1 Validation

Software Version: R7-7.8.0

Date: May 13th, 2022

Title: Oracle Linux 7 NSS Cryptographic Module Security Policy
Date: May 13th, 2022
Author: Oracle Security Evaluations – Global Product Security
Contributing Authors:
Oracle Linux Engineering
atsec information security

Oracle Corporation
World Headquarters
2300 Oracle Way
Austin, TX 78741
U.S.A.
Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

ORACLE

TABLE OF CONTENTS

List of Tables

List of Figures

1. Introduction

1.1 Overview

This document is the Security Policy for the Oracle Linux 7 NSS Cryptographic Module by Oracle Corporation. This Security Policy specifies the security rules under which the module shall operate to meet the requirements of FIPS 140-2 Level 1. It also describes how the Oracle Linux 7 NSS Cryptographic Module functions in order to meet the FIPS 140-2 requirements, and the actions that operators must take to maintain the security of the module.

This Security Policy describes the features and design of the Oracle Linux 7 NSS Cryptographic Module using the terminology contained in the FIPS 140-2 specification. FIPS 140-2, Security Requirements for Cryptographic Module specifies the security requirements that will be satisfied by a cryptographic module utilized within a security system protecting sensitive but unclassified information. The NIST/CCCS Cryptographic Module Validation Program (CMVP) validates cryptographic module to FIPS 140-2. Validated products are accepted by the Federal agencies of both the USA and Canada for the protection of sensitive or designated information.

1.2 Document Organization

The FIPS 140-2 Submission Package contains:

- Oracle Linux 7 NSS Cryptographic Module Non-Proprietary Security Policy
- Other supporting documentation as additional references

With the exception of this Non-Proprietary Security Policy, the FIPS 140-2 Validation Documentation is proprietary to Oracle and is releasable only under appropriate non-disclosure agreements. For access to these documents, please contact Oracle.

2. Oracle Linux 7 NSS Cryptographic Module

2.1 Functional Overview

The Oracle Linux 7 NSS Cryptographic Module (hereafter referred to as the "module") is a set of libraries designed to support cross-platform development of security-enabled applications. Applications built with the Oracle Linux 7 NSS Cryptographic Module can support TLS, PKCS #5, PKCS #7, PKCS #11 (version 3.0), PKCS #12, S/MIME, X.509 v3 certificates, and other security standards supporting FIPS 140-2 validated cryptographic algorithms. It combines a vertical stack of Oracle Linux components intended to limit the external interface each separate component may provide. The Oracle Linux 7 NSS Cryptographic Module is distributed with the Oracle Linux open-source distributions. The module provides a C-language Application Program Interface (API) for use by other processes that require cryptographic functionality.

Oracle Linux 7 NSS Cryptographic Module supports 3 types of cryptographic implementations:

a) NSS in Native C Programming Language; and
b) AES-NI for X86 processors.
c) AES optimizations for ARM processors

2.2 FIPS 140-2 Validation Scope

The following table shows the security level for each of the eleven sections of the validation. See Table 1 below.

Security Requirements Section	Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles and Services and Authentication	2
Finite State Machine 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

Table 1: FIPS 140-2 Security Requirements

3. Cryptographic Module Specification

3.1 Definition of the Cryptographic Module

The Oracle Linux 7 NSS Cryptographic Module with version R7-7.8.0 is defined as a software only multi-chip standalone module as defined by the requirements within FIPS PUB 140-2. The logical cryptographic boundary of the module consists of shared library files and their integrity check signature files, which are delivered through the Package Manager (RPM) as listed below:

- nss-softokn RPM file with version nss-softokn-3.53.1-6.0.1.el7_9.x86_64.rpm or nss-softokn-3.53.1-6.0.1.el7_9.aarch64.rpm which contains the following files:
  - /usr/lib64/libnssdbm3.chk (64 bits)
  - /usr/lib64/libnssdbm3.so (64 bits)
  - /usr/lib64/libsoftokn3.chk (64 bits)
  - /usr/lib64/libsoftokn3.so (64 bits)

- nss-softokn-freebl RPM file with version nss-softokn-freebl-3.53.1-6.0.1.el7_9.x86_64.rpm or nss-softokn-freebl-3.53.1-6.0.1.el7_9.aarch64.rpm, which contains the following files:
  - /lib64/libfreeblpriv3.chk (64 bits)
  - /lib64/libfreeblpriv3.so (64 bits)

Figure 1 shows the logical block diagram of the module executing in memory on the host system.

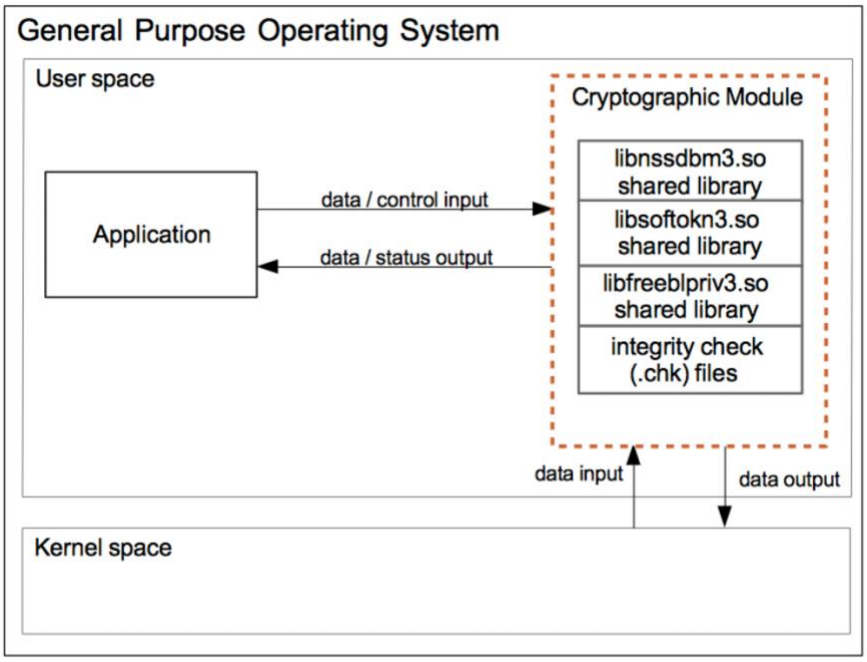

Figure 1: Oracle Linux 7 NSS Logical Cryptographic Boundary

3.2 Definition of the Physical Cryptographic Boundary

The physical cryptographic boundary of the module is defined as the hard enclosure of the host system on which it runs. See Figure 2 below. No components are excluded from the requirements of FIPS PUB 140-2.

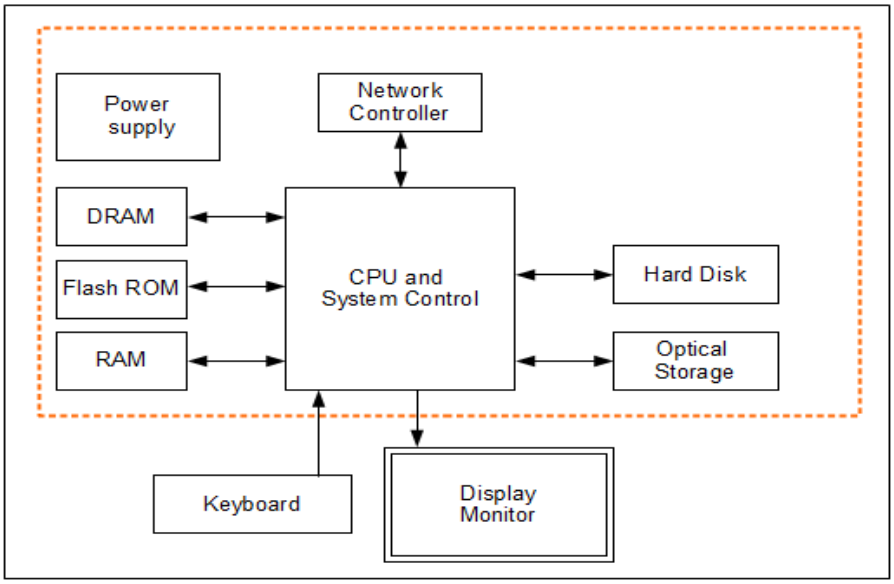

Figure 2: Oracle Linux 7 NSS Hardware Block Diagram

3.3 Approved or Allowed Security Functions

The module supports two modes of operation: FIPS Approved mode and non-Approved mode.

When the module is powered on, the power-up self-tests are executed automatically without any operator intervention. If the power-up self-tests complete successfully, the module will be in FIPS Approved mode by default. In Approved mode, only Approved algorithms (as listed in Table 2) and non-approved but Allowed algorithms (as listed in Table 3) can be used.

Approved Security Functions		Certificate
Symmetric Algorithms		
AES	**Generic C:** AES in CBC, ECB, CTR, GCM Modes (e/d; Key Sizes 128, 192, 256 for all modes)	A 1179
	AESNI: AES in CBC, ECB, CTR, GCM Modes (e/d; Key Sizes 128, 192, 256 for all modes)	A 1097
	Generic C CTS: AES in CBC-CS1 mode (d/e; Key Sizes 128, 192, 256)	A 1183
	Generic C CMAC: Generation/Verification (Key Sizes 128, 192, 256) MAC: 128	A 1060
	AESNI_KW: AES-KW (d/e; Key Sizes 128, 192, 256) AES-KWP (d/e; Key Sizes 128, 192, 256)	A 1072
	Generic C KW: AES-KW (d/e; Key Sizes 128, 192, 256) AES-KWP (d/e; Key Sizes 128, 192, 256)	A 1059

Approved Security Functions		Certificate
	CE_KW AES-KW (d/e; Key Sizes 128, 192, 256) AES-KWP (d/e; Key Sizes 128, 192, 256)	A 2579
	CE AES-CBC, AES-ECB, AES-GCM (d/e; Key Sizes 128, 192, 256)	A 2580
Triple-DES (3-Key)[1]	**Generic C:** Triple-DES in CBC and ECB Modes (e/d; KO 1)	A 1179
Cryptographic Key Generation (CKG)		
CKG	**NIST SP 800-133rev2** Asymmetric and Symmetric Cryptographic Key Generation	(Vendor Affirmed)
Secure Hash Standard (SHS)		
SHS	**Generic_C:** SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	A 1179
Data Authentication Code		
HMAC	**Generic_C:** HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	A 1179
Asymmetric Algorithms		
RSA	**Generic_C:** **FIPS 186-4:** PKCS 1.5 (Key Gen, Sig Gen, Sig Ver); Modulus Sizes 2048, 3072, 4096 with hash sizes SHA-256, SHA-384, SHA-512; (SHA-1 allowed for Sig Ver operations supported)	A 1179
DSA	**Generic_C:** **FIPS 186-4:** Key Gen, PQG Gen, PQG Ver, Sig Gen, Sig Ver; Modulus Sizes 2048, 3072, with hash sizes SHA-224, SHA-256, SHA-384, SHA-512; (Modulus size 1024 and SHA-1 allowed for PQG Ver and Sig Ver operations only)	A 1179
ECDSA	**Generic_C:** **FIPS 186-4:** Key Gen, Key Ver, Sig Gen, Sig Ver; Curves P-256, P-384, P-521 with hash sizes SHA-224, SHA-256, SHA-384, SHA-512; (SHA-1 allowed for Sig Ver operations only)	A 1179
Random Number Generation		
DRBG	**Generic_C:** Hash_Based DRBG: (SHA-256)	A 1179
Key Agreement Scheme (NIST SP 800-56Ar3)		
KAS-FFC and KAS-FFC-SSC-SP800-56Ar3	**Generic_C:** **KAS-FFC Component:** dhEphem scheme, Full Public Key Validation; (SHA-224, SHA-256) **KAS-FFC-SSC SP 800-56Ar3:** dhEphem scheme, Domain Parameter Generation and Mod P Methods (ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192) KAS role: initiator, responder	A 1179

[1] 3-Key Triple-DES key shall not be used to encrypt more than 2^{16} 64-bit blocks of data.

Approved Security Functions		Certificate
KAS-ECC and KAS-ECC-SSC-SP800-56Ar3	**Generic_C:** **KAS-ECC Component:** Partial Public Key Validation; (Curves P-256 with SHA-256, P-384 with SHA-384, P-521 with SHA-521) **KAS-ECC-SSC SP 800-56Ar3:** ephemeralUnified scheme for EC Diffie-Hellman shared secret computation (Curves P-256, P-384, P-521). KAS role: initiator, responder	
Safe Primes Key Generation	**Generic_C:** **Safe Primes Key Generation:** Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	
Key Derivation		
KDF IKE (CVL)	**IKE_KDF:** IKEv1, IKEv2 (SHA 1, 256, 384, 512)	A 1061
KDF TLS (CVL)	**Generic_C:** TLS 1.0, TLS 1.1, TLS 1.2 with SHA-256, SHA-384, SHA-512	A 1179
KDF HKDF (KDA)	**HKDF 56Cr1 to support the TLS 1.3 PRF:** HKDF: (NIST SP 800-56Cr1 with HMAC-SHA (224, 256, 384, 512))	A 1071
PBKDF2	**Generic_C:** PBKDF2 with HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	A 1179
Key Transport Scheme (KTS)		
KTS	**AESNI_KW:** AES-KW (d/e; Key Sizes 128, 192, 256) AES-KWP (d/e; Key Sizes 128, 192, 256)	A 1072
	Generic C KW: AES-KW (d/e; Key Sizes 128, 192, 256) AES-KWP (d/e; Key Sizes 128, 192, 256)	A 1059
	CE_KW AES-KW (d/e; Key Sizes 128, 192, 256) AES-KWP (d/e; Key Sizes 128, 192, 256)	A 2579
	CE AES-GCM key wrapping with 128, 192 and 256 bit keys	A 2580
	Generic C AES-GCM key wrapping with 128, 192 and 256 bit keys	A 1179
	AESNI AES-GCM key wrapping with 128, 192 and 256 bit keys	A 1097
	AES-CBC with 128 and 256 bit keys and HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, or HMAC-SHA-512 key wrapping	A 1097 (AES) A 2580 (AES)
		A 1179 (AES & HMAC)

Approved Security Functions		Certificate
	Triple-DES CBC with 192[2] bit key and HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, or HMAC-SHA-512 key wrapping	A 1179
Entropy		
ENT(NP)	**NIST SP 800-90B**	N/A

Table 2: FIPS Approved Security Functions

Note: No parts of the TLS protocol except the KDF with cert #A 1179 and #A 1071 have been reviewed or tested by CMVP or CAVP. Also, no parts of the IKE protocol except the KDF with cert #A 1061 have been reviewed or tested by CMVP or CAVP.

3.4 Non-Approved but Allowed Security Functions

The following are considered non-Approved but allowed security functions:

Algorithm	Usage
RSA PKCS#1-v1.5 Key Wrapping with key sizes greater than 2048-bit up to 16384 bits	Key wrapping using PKCS#1-v1.5 padding scheme method, key establishment methodology provides between 112 and 256 bits of encryption strength.
MD5 (no security claimed) Allowed to be used in FIPS mode per IG 1.23	Message digest used in TLS only.
PBKDF1 key derivation	Used as part of User authentication mechanism[3]. Allowed to be used in FIPS mode per IG 1.23 (no security claimed)
Two key Triple-DES encrypt/decrypt	Used as part of User authentication mechanism[3]. Allowed to be used in FIPS mode per IG 1.23 (no security claimed)

Table 3: Non-Approved but Allowed Security Functions

3.5 Non-Approved Security Functions

The following algorithms are considered non-Approved. Using any of these algorithms will put the module in the non-Approved mode implicitly. The services associated with these non-Approved algorithms are specified in Section 7.3.

Algorithm	Usage
Camellia	Encryption/Decryption
ChaCha20-Poly1305	Authenticated Encryption/Decryption
DES	Encryption/Decryption
RC2	Encryption/Decryption
RC4	Encryption/Decryption

[2] Though the key size is 192 bits, the strength of that key and therefore the KTS implementation for Triple-DES is 112 bits only according to IG 7.5.

[3] Please note that these algorithms are only used as legacy user authentication. For the newly created users, the PBKDF2 and AES CBC algorithms are used.

Algorithm	Usage
RC5	Encryption/Decryption
SEED	Encryption/Decryption
Two-Key Triple-DES	Encryption/Decryption, key wrapping using two-key Triple-DES
MD2	Hashing
MD5	Hashing
DSA with non-compliant key size	DSA key pair generation with key size less than 2048 bits.
	DSA signature generation with key size less than 2048 bits.
	DSA signature generation with SHA-1.
	DSA signature verification with key size less than 1024 bits.
RSA with non-compliant key size	RSA key pair generation with key size less than 2048 bits..
	RSA Signature generation with SHA-1
	RSA signature generation with key size less than 2048 bits
	RSA signature verification with key size less than 1024 bits.
	RSA PKCS#1-v1.5 key wrapping with key size less than 2048 bits.
RSA PSS (non-compliant)	Signature Generation/Verification
RSA OAEP (non-compliant)	Key wrapping
ECDSA with non-compliant hash size	ECDSA signature generation using SHA-1
Diffie-Hellman with non-compliant key size	Diffie-Hellman key agreement or shared secret computation with key size less than 2048 bits.
Diffie-Hellman keys generated with domain parameters other than safe primes.	Key agreement, Shared Secret computation
EC Diffie-Hellman with P-192 curve, K curves, B curves and non-NIST curves.	Key agreement, Shared Secret computation
J-PAKE	Key agreement
Key Wrapping	non-SP 800-38F AES and Triple-DES Key wrapping

Table 4: Non-Approved and Disallowed Functions

4. Module Ports and Interfaces

The module FIPS 140 interfaces can be categorized as follows:

- Data Input Interface
- Data Output Interface
- Control Input interface
- Status Output Interface

As a software-only module, the module does not have physical ports. For the purpose of FIPS 140-2 validation, the physical ports of the module are interpreted to be the physical ports of the hardware platform on which it runs. The logical interface is a C-language Application Program Interface (API) following the PKCS #11 specification, the database files in kernel file system, and environment variables.

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 entering the module and data/status exiting the module. The module doesn't 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.

Table 5 below shows the mapping of interfaces as per FIPS 140-2 standard.

FIPS 140-2 Interface	Module Interfaces
Data Input	API input parameters and database files in kernel file system
Data Output	API output parameters and database files in kernel file system
Control Input	API function calls and environment variables
Status Output	API return codes and status parameters

Table 5: Mapping of FIPS 140-2 Logical Interfaces

4.1 PKCS#11

The logical interfaces of the module consist of the PKCS #11 (Cryptoki) API. The API itself defines the module's logical boundary, i.e., all access to the module is through this API. The functions in the PKCS #11 API are listed in Table 8.

4.2 Inhibition of Data Output

All data output via the data output interface is inhibited when the NSS cryptographic 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 NSS cryptographic 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.

4.3 Disconnecting the Output Data Path from the Key Processes

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.

5. Physical Security

The module is comprised of software only and thus does not claim any physical security.

6. Operational Environment

6.1 Tested Environments

The module operates in a modifiable operational environment under the FIPS 140-2 definition. The module was tested on the following environments with and without PAA (i.e., AES-NI and CE):

Operating Environment	Processor	Hardware
Oracle Linux 7.8 64-bit	Intel® Xeon® Platinum 8167M	Oracle Server X7-2C
Oracle Linux 7.8 64-bit	AMD EPYC™ 7551	Oracle Server E1-2C
Oracle Linux 7.8 64-bit	Ampere®Altra® Neoverse-N1	Oracle Server A1-2C

Table 6: Tested Operating Environment

6.2 Vendor Affirmed Environments

The following platforms have not been tested as part of the FIPS 140-2 level 1 certification. However, Oracle "vendor affirms" that these platforms are equivalent to the tested and validated platforms. Additionally, Oracle affirms that the module will function the same way and provide the same security services on any of the systems listed below.

Operating Environment	Hardware
Oracle Linux 7 64-bit	Oracle X Series Servers
Oracle Linux 7 64-bit	Oracle E Series Servers
Oracle Linux 7 64-bit	Oracle A Series Servers

Table 7: Vendor Affirmed Operational Environments

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

6.3 Operational Environment Policy

The operating system is restricted to a single operator (concurrent operators are explicitly excluded).

The application that makes calls to the module is the single user of the module, even when the application is serving multiple clients.

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

7. Roles, Services and Authentication

This section defines the roles, services, and authentication mechanisms and methods with respect to the applicable FIPS 140-2 requirements.

7.1 Roles

The module implements a Crypto Officer (CO) role and a User role:

- The CO role is supported for the installation and initialization of the module. Also, the CO role 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 management of physical access to the computer, executing the module code as well as management of the security facilities provided by the operating system.
- 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.

7.2 FIPS Approved Operator Services and Descriptions

The module has a set of API functions denoted by FC_xxx as listed in Table 8. 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. User services which access CSPs (e.g., FC_GenerateKey, FC_GenerateKeyPair) require operator authentication.

Table 8 lists all the services available in FIPS Approved mode. Please refer to Table 2 and Table 3 for the Approved or allowed cryptographic algorithms supported by the module.

U	CO	Service Name	Function	Service Description	Keys and CSP(s)	Access
	X	Get Function List	FC_GetFunctionList	Return a pointer to the list of function pointers for the operational mode	None	-
	X	Module Initialization	FC_InitToken	Initialize or re-initialize a token	User password and all keys	Z
	X		FC_InitPIN	Initialize the user's password, i.e., set the user's initial password	User password	W
	X	General Purpose	FC_Initialize	Initialize the module library	None	-
	X		FC_Finalize	Finalize (shut down) the module library	All keys	Z
	X		FC_GetInfo	Obtain general information about the module library	None	-
	X	Slot and Token Management	FC_GetSlotList	Obtain a list of slots in the system	None	-
	X		FC_GetSlotInfo	Obtain information about a particular slot	None	-
	X		FC_GetTokenInfo	Obtain information about the token (This function provides the Show Status service)	None	-
	X		FC_GetMechanismList	Obtain a list of mechanisms (cryptographic algorithms) supported by a token	None	-
	X		FC_GetMechanismInfo	Obtain information about a particular mechanism	None	-
X			FC_SetPIN	Change the user's password	User password	R, W
	X	Session Management	FC_OpenSession	Open a connection (session) between an application and a particular token	None	-
	X		FC_CloseSession	Close a session	All keys for the session	Z
	X		FC_CloseAllSessions	Close all sessions with a token	All keys	Z
	X		FC_GetSessionInfo	Obtain information about the session (This function provides the Show Status service)	None	-
	X		FC_GetOperationState	Save the state of the cryptographic operations in a session (This function is only implemented for message digest operations)	None	-
	X		FC_SetOperationState	Restore the state of the cryptographic operations in a session (This function is only implemented for message digest operations)	None	-
X			FC_Login	Log into a token	User Password	R, W, X
X			FC_Logout	Log out from a token	None	-
X		Object Management	FC_CreateObject	Create a new object	Key	W

U	CO	Service Name	Function	Service Description	Keys and CSP(s)	Access
X			FC_CopyObject	Create a copy of an object	Original Key, new key[4]	R, W
X			FC_DestroyObject	Destroy an object	Key	Z
X			FC_GetObjectSize	Obtain the size of an object in bytes	Key	R
X			FC_GetAttributeValue	Obtain an attribute value of an object	Key	R
X			FC_SetAttributeValue	Modify an attribute value of an object	Key	W
X			FC_FindObjectsInit	Initialize an object search operation	None	-
X			FC_FindObjects	Continue an object search operation	Keys matching the search criteria	R
X			FC_FindObjectsFinal	Finish an object search operation	None	-
X		Encryption and Decryption	FC_EncryptInit	Initialize an encryption operation	AES/Triple-DES key	R
X			FC_Encrypt	Encrypt single-part data	AES/Triple-DES key	R
X			FC_EncryptUpdate	Continue a multiple-part encryption operation	AES/Triple-DES key	R
X			FC_EncryptFinal	Finish a multiple-part encryption operation	AES/Triple-DES key	R
X			FC_DecryptInit	Initialize a decryption operation	AES/Triple-DES key	R
X			FC_Decrypt	Decrypt single-part encrypted data	AES/Triple-DES key	R
X			FC_DecryptUpdate	Continue a multiple-part decryption operation	AES/Triple-DES key	R
X			FC_DecryptFinal	Finish a multiple-part decryption operation	AES/Triple-DES key	R
	X	Message Digest	FC_DigestInit	Initialize a message digesting operation	None	-
	X		FC_Digest	Digest single-part data	None	-
	X		FC_DigestUpdate	Continue a multiple-part digesting operation	None	-
X			FC_DigestKey	Continue a multiple-part message-digesting operation by digesting the value of a secret key as part of the data already digested	HMAC Key	R
	X		FC_DigestFinal	Finish a multiple-part digesting operation	None	-
X		Signature Generation and Verification	FC_SignInit	Initialize a signature operation	DSA/ECDSA/RSA private key, HMAC key	R
X			FC_Sign	Sign single-part data	DSA/ECDSA/RSA private key, HMAC key	R
X			FC_SignUpdate	Continue a multiple-part signature operation	DSA/ECDSA/RSA private key, HMAC key	R

[4] 'Original key' and 'New key' are the secret keys or public/private key pairs.

U	CO	Service Name	Function	Service Description	Keys and CSP(s)	Access
X			FC_SignFinal	Finish a multiple-part signature operation	DSA/ECDSA/RSA private key, HMAC key	R
X			FC_SignRecoverInit	Initialize a signature operation, where the data can be recovered from the signature	DSA/ECDSA/RSA private key	R
X			FC_SignRecover	Sign single-part data, where the data can be recovered from the signature	DSA/ECDSA/RSA private key	R
X			FC_VerifyInit	Initialize a verification operation	DSA/ECDSA/RSA public key, HMAC key	R
X			FC_Verify	Verify a signature on single-part data	DSA/ECDSA/RSA public key, HMAC key	R
X			FC_VerifyUpdate	Continue a multiple-part verification operation	DSA/ECDSA/RSA public key, HMAC key	R
X			FC_VerifyFinal	Finish a multiple-part verification operation	DSA/ECDSA/RSA public key, HMAC key	R
X			FC_VerifyRecoverInit	Initialize a verification operation, where the data is recovered from the signature	DSA/ECDSA/RSA public key	R
X			FC_VerifyRecover	Verify a signature on single-part data, where the data is recovered from the signature	DSA/ECDSA/RSA public key	R
X		Dual Function Cryptographic Operations	FC_DigestEncryptUpdate	Continue a multiple-part digesting and encryption operation	AES/Triple-DES key	R
X			FC_DecryptDigestUpate	Continue a multiple-part decryption and digesting operation	AES/Triple-DES key	R
X			FC_SignEncryptUpdate	Continue a multiple-part signing and encryption operation	DSA/ECDSA/RSA private key, HMAC key, AES/Triple-DES key	R
X			FC_DecryptVerifyUpdate	Continue a multiple-part decryption and verify operation	DSA/ECDSA/RSA public key, HMAC key, AES/Triple-DES key	R
X		Key Management	FC_GenerateKey	Generate a secret key (Also used by TLS to generate a pre-master secret)	AES/Triple-DES/HMAC key	W
				Derive a key from an input password or passphrase using PBKDF2	PBKDF2 password or passphrase	R
					PBKDF2 derived key	W
			FC_GenerateKeyPair	Generate a public/private key pair	DSA/ECDSA/RSA key pair, Diffie-Hellman/EC Diffie-Hellman key pair	W

U	CO	Service Name	Function	Service Description	Keys and CSP(s)	Access
			FC_WrapKey	Wrap (encrypt) a key using one of the following mechanisms: (1) SP 800-38F AES Key wrapping (2) RSA encryption using PKCS#1-v1.5	Wrapping Key[5], Key to be wrapped[6]	R
			FC_UnwrapKey	Unwrap (decrypt) a key using one of the following mechanisms: (1) SP 800-38F AES Key unwrapping (2) RSA decryption using PKCS#1-v1.5	Unwrapping key[7]	R
					Unwrapped key[8]	W
			FC_DeriveKey	Shared secret computation	Diffie-Hellman and EC Diffie-Hellman key pairs	R
					Diffie-Hellman and EC Diffie-Hellman shared secrets	W
				TLS derived key from TLS master secret which is derived from TLS premaster secret using TLS KDF	TLS pre-master secret	R
					TLS master secret	R, W
					TLS derived key[9]	W
				Derive keys from IKE shared secret using IKE KDF.	IKE shared secret	R
					IKE derived keys	W
	X	Random Number Generation	FC_SeedRandom	Mix in additional seed material to the random number generator	Entropy input string, seed, DRBG V and C values	R, W
			FC_GenerateRandom	Generate random data (This function performs the continuous random number generator test)	Random data, DRBG V and C values	R, W
	X	Parallel Function Management	FC_GetFunctionStatus	A legacy function, which simply returns the value 0x00000051 (function not parallel)	None	-
			FC_CancelFunction	A legacy function, which simply returns the value 0x00000051 (function not parallel)	None	-
	X	Self-Tests	N/A	The self-tests are performed automatically when loading the module	DSA 2048-bit public key for module integrity test	R
	X	Show Status	N/A	Via exit codes	N/A	N/A

[5] 'Wrapping key' corresponds to the secret key or public key used to wrap another key.
[6] 'Key to be wrapped' is the key that is wrapped by the 'wrapping key'.
[7] 'Unwrapping key' corresponds to the secret key or private key used to unwrap another key.
[8] 'Unwrapped key' is the plaintext key that has not been wrapped by a 'wrapping key'.
[9] 'Derived key' is the key obtained by a key derivation function which takes the 'TLS master secret' as input.

U	CO	Service Name	Function	Service Description	Keys and CSP(s)	Access
X		Zeroization	FC_DestroyObject	All CSPs are automatically zeroized when freeing the cipher handle	All secret or private keys and PBKDF2 password	Z
	X		FC_InitToken FC_Finalize FC_CloseSession FC_CloseAllSessions			

Table 8: FIPS Approved Operator Services and Descriptions

R – Read, W – Write, X – Execute, Z – Zeroize

Note: 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.

7.3 Non-FIPS Approved Services and Descriptions

Table 9 lists all the services available in non-Approved mode with API function and the non- Approved algorithm that the function may invoke. Please note that the functions are the same as the ones listed in **Table 8**, but the underneath non-Approved algorithms 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.

Service Name	Function	Non-Approved Algorithm Invoked
Encryption and Decryption	FC_EncryptInit	Camellia, ChaCha20-Poly 1305 (authenticated), DES, RC2, RC4, RC5, SEED, Two-Key Triple-DES
	FC_Encrypt	
	FC_EncryptUpdate	
	FC_EncryptFinal	
	FC_DecryptInit	Camellia, ChaCha20-Poly 1305 (authenticated), DES, RC2, RC4, RC5, SEED, Two-Key Triple-DES
	FC_Decrypt	
	FC_DecryptUpdate	
	FC_DecryptFinal	
Message Digest	FC_DigestInit	MD2, MD5
	FC_Digest	
	FC_DigestUpdate	
	FC_DigestKey	
	FC_DigestFinal	
	FC_SignInit	

Service Name	Function	Non-Approved Algorithm Invoked
Signature Generation and Verification	FC_Sign	DSA signature generation with non-compliant key size < 2048, DSA signature generation with SHA-1, RSA PSS (non-compliant), RSA signature generation with non-compliant key sizes < 2048, RSA signature generation with SHA-1, ECDSA signature generation using SHA-1
	FC_SignUpdate	
	FC_SignFinal	
	FC_SignRecoverInit	
	FC_SignRecover	
	FC_VerifyInit	DSA signature verification with non-compliant key sizes < 1024, RSA PSS (non-compliant), RSA signature verification with non-compliant key sizes < 1024
	FC_Verify	
	FC_VerifyUpdate	
	FC_VerifyFinal	
	FC_VerifyRecoverInit	
	FC_VerifyRecover	
Dual Function Cryptographic Operations	FC_DigestEncryptUpdate	MD2, MD5, Camellia, DES, RC2, RC4, RC5, SEED, Two-Key Triple-DES
	FC_DecryptDigestUpdate	Camellia, DES, RC2, RC4, RC5, SEED, MD2, MD5
	FC_SignEncryptUpdate	DSA signature generation with non-compliant key sizes < 2048, DSA signature generation with SHA-1, RSA PSS (non-compliant), RSA signature generation with non-compliant key sizes < 2048, RSA signature generation with SHA-1, ECDSA signature generation with SHA-1, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
	FC_DecryptVerifyUpdate	DSA signature verification with non-compliant key sizes < 1024, RSA PSS (non-compliant), RSA signature verification with non-compliant key sizes < 1024, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
Key Management	FC_GenerateKeyPair	DSA key pair generation with non-compliant key sizes < 2048, RSA key pair generation with non-compliant key sizes < 2048
	FC_WrapKey	Triple-DES key wrapping (encrypt) using Two-key, Non-SP 800-38F Triple-DES key wrapping (encrypt) using Three-key Triple-DES, RSA key wrapping (encrypt) using PKCS#1-v1.5 with non-compliant key sizes < 2048, RSA key wrapping (encrypt) using OAEP, non-SP 800-38F AES key wrapping (encrypt)
	FC_UnwrapKey	Triple-DES key wrapping (decrypt) using Two-key, Non-SP 800-38F Triple-DES key wrapping (decrypt) using Three-key Triple-DES, RSA key wrapping (decrypt) using PKCS#1-v1.5 with non-compliant key sizes < 2048, RSA key wrapping (decrypt) using OAEP, non-SP 800-38F AES key unwrapping (decrypt)
	FC_DeriveKey	Diffie-Hellman key agreement or shared secret computation with noncompliant key sizes < 2048, J-PAKE key agreement
		Diffie-Hellman keys generated with domain parameters other than safe primes.
		EC Diffie-Hellman with P-192 curve, K curves, B curves and non-NIST curves.

Table 9: Non-FIPS Approved Operator Services and Descriptions

7.4 Operator Authentication

7.4.1 Role Assumption

The CO role is implicitly assumed by an operator while installing the module by following the instructions in Section 10.1 and while performing other CO services on the module.

The module implements a user password-based Role based authentication for the User role as defined by FIPS 140-2. To perform any security services under the User role, an operator must log into the module and complete an authentication procedure using the user password information unique to the User role operator. The user password is passed to the module via the API function as an input argument and won't 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 user password is initialized by the CO role as part of module initialization and can be changed by the User role operator.

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

Once a user 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. User password establishment and authentication are required for the operation of the module. When the module is powered off, the result of previous authentication will be cleared and the user needs to be re-authenticated.

7.4.2 Strength of Authentication Mechanism

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

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

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

- The characters of the user password are independent with each other.

- The user password contains the smallest combination of the character classes, which are 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 user password can contain seven characters from any three or more of the aforementioned five character classes, the probability that a random guess of the user password will succeed is less than or equals to 1/67,600,000, which is smaller than the required threshold 1/1,000,000.

After each failed authentication attempt, the NSS cryptographic 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 user password during a one-minute period is less than or equals to 60 * 1/67,600,000 = 0.089 * (1/100,000), which is smaller than the required threshold 1/100,000.

8. Key and CSP Management

The following keys, cryptographic key components and other critical security parameters are contained in the module.

CSP Name	Generation	Entry/Output	Storage	Zeroization
AES Key (128, 192, 256 bits)	NIST SP 800-90A DRBG	Encrypted through key wrapping using FC_WrapKey	Application memory or key database	Automatically zeroized when freeing the cipher handle
Triple-DES Key (192 bits)[10]	NIST SP 800-90A DRBG	Encrypted through key wrapping using FC_WrapKey	Application memory or key database	Automatically zeroized when freeing the cipher handle
DSA Private Key (2048 and 3072 bits)	Public and private keys are generated using the FIPS 186-4 key generation method via the FC_GenerateKeyPair function; random values are obtained from the SP800-90A DRBG.	Encrypted through key wrapping using FC_WrapKey	Application memory or key database	Automatically zeroized when freeing the cipher handle
ECDSA Private Key (P-256, P-384, P-521)	Public and private keys are generated using the FIPS 186-4 key generation method via the FC_GenerateKeyPair function; random values are obtained from the SP800-90A DRBG.	Encrypted through key wrapping using FC_WrapKey	Application memory or key database	Automatically zeroized when freeing the cipher handle
RSA Private Key (2048, 3072, 4096 bits)	Public and private keys are generated using the FIPS 186-4 key generation method via the FC_GenerateKeyPair function; random values are obtained	Encrypted through key wrapping using FC_WrapKey	Application memory or key database	Automatically zeroized when freeing the cipher handle

[10] Triple-DES key length is 192 bits consisting of 168 security relevant bits and 24 parity bits. The encryption strength is 112 bits as outlined in IG 7.5.

CSP Name	Generation	Entry/Output	Storage	Zeroization
	from the SP800-90A DRBG.			
HMAC Key (≥ 112 bits)	NIST SP 800-90A DRBG	Encrypted through key wrapping using FC_WrapKey	Application memory or key database	Automatically zeroized when freeing the cipher handle
DRBG Entropy Input String	Obtained from CPU jitter source	N/A	Application memory	Automatically zeroized when freeing DRBG handle
DRBG seed, V and C values	Derived from entropy input	N/A	Application memory	Automatically zeroized when freeing DRBG handle
TLS Pre-Master Secret	Use of SP 800-90A DRBG or DH/ECDH key agreement	N/A	Application memory or key database	Automatically zeroized when freeing the cipher handle
TLS Master Secret	Derived from TLS pre-master secret	N/A	Application memory or key database	Automatically zeroized when freeing the cipher handle
TLS derived keys	TLS derived keys Generated during the TLS v1.0/1.1 and v1.2 KDFs from TLS master-secret	Keys are output in plaintext form.	Application memory or key database	Automatically zeroized when freeing the cipher handle
Diffie-Hellman public and private keys	Public and private keys are generated using the SP800-56ARev3 Safe Primes key generation method via the FC_GenerateKeyPair function; random values are obtained from the SP800-90A DRBG.	Encrypted through key wrapping using FC_WrapKey	Application memory or key database	Automatically zeroized when freeing the cipher handle
EC Diffie-Hellman public and private keys	Public and private keys are generated using the FIPS 186-4 key generation method via the FC_GenerateKeyPair function; random values are obtained	Encrypted through key wrapping using FC_WrapKey	Application memory, key storage	Automatically zeroized when freeing the cipher handle

CSP Name	Generation	Entry/Output	Storage	Zeroization
	from the SP800-90A DRBG.			
Diffie-Hellman shared secret	Generated during shared secret computation via FC_DeriveKey.	Shared secret is output in plaintext form.	Application memory, key storage	Automatically zeroized when freeing the cipher handle
EC Diffie-Hellman shared secret	Generated during shared secret computation via FC_DeriveKey.	Shared secret is output in plaintext form.	Application memory, key storage	Automatically zeroized when freeing the cipher handle
PBKDF2 password	N/A	API input parameter	Application memory or key database	Automatically zeroized when freeing the cipher handle
PBKDF2 derived key	Derived using SP800-132 PBKDF2 mechanisms	Encrypted for output using FC_WrapKey	Application memory or key database	Automatically zeroized when freeing the cipher handle
IKE shared secret	Provided by calling application.	The IKE shared secret is output in plaintext form.	Application memory or key database	Automatically zeroized when freeing the cipher handle
IKE derived keys	Generated during the IKEv1 and IKEv2 KDFs	Keys are output in plaintext form.	Application memory or key database	Automatically zeroized when freeing the cipher handle
User password	N/A (supplied by the calling application)	N/A (input through API parameter)	Application memory or key database in salted form	Automatically zeroized when the module is reinitialized or overwritten when the user changes its user password

Table 10: CSP Table

8.1 Random Number Generation

The module employs the Deterministic Random Bit Generator (DRBG) based on [SP 800-90A] for the random number generation. The DRBG supports the Hash_DRBG mechanism with SHA-256. The module uses CPU jitter as a noise source provided by the operational environment which is within the module's physical boundary but outside of the module's logical boundary. The source is compliant with [SP 800-90B] and marked as ENT on the certificate. The entropy source provides at least 223 bits of entropy and the following caveat applies: The module generates keys whose strength are modified by available entropy. Lastly, the module performs the DRBG health tests as defined in section 11.3 of [SP 800-90A].

8.2 Key/CSP Storage

The module employs the cryptographic keys and CSPs in the FIPS Approved mode of operation as listed in Table 10. 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) mentioned in Table 10 is within the module's physical boundary but outside its logical boundary.

8.3 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 CSP. A plaintext secret or private key gets zeroized when it is passed to an FC_DestroyObject call. All plaintext secret and private keys must be zeroized when the module is shut down (with an FC_Finalize call), reinitialized (with an FC_InitToken call), or when the session is closed (with an FC_CloseSession or FC_CloseAllSessions call). All zeroization is to be performed by storing the value 0 into every byte of the memory region that is previously occupied by a plaintext secret key, private key or CSP. Zeroization is performed in a time that is not sufficient to compromise plaintext secret or private keys and CSP.

8.4 Key/CSP Generation

The key generation methods implemented in the module in the Approved mode are compliant with [SP 800-133].

The module generates symmetric key using the unmodified output of SP 800-90A DRBG per SP 800-133rev2 section 6.1. Symmetric keys can also be derived from KDF as stated in section 8.6.

For generating RSA, DSA and ECDSA keys the module implements asymmetric key generation services compliant with [FIPS 186-4]. A seed (i.e. the random value) used in asymmetric key generation is directly obtained from the [SP 800-90A] DRBG.

The public and private keys used in the EC Diffie-Hellman key agreement schemes are generated internally by the module using ECDSA key generation compliant with [FIPS 186-4] and [SP 800-56ARev3]. The Diffie-Hellman key agreement scheme is also compliant with [SP 800-56ARev3], and generates keys using safe primes defined in RFC 7919 and RFC 3526, as described in the next section.

8.5 Key Agreement/Key Transport

The vendor documentation states the module provides Diffie-Hellman shared secret computation (KAS FFC SSC) and EC Diffie-Hellman shared secret computation (KAS ECC SSC) compliant with SP 800-56Arev3, in accordance with scenario X1 (1) of IG D.8.

For Diffie-Hellman, the module supports the use of safe primes defined in RFC 7919 for domain parameters and key generation, which are used in TLS key exchange.

- TLS (RFC7919)
  - ffdhe2048 (ID = 256)
  - ffdhe3072 (ID = 257)
  - ffdhe4096 (ID = 258)
  - ffdhe6144 (ID = 259)
  - ffdhe8192 (ID = 260)

The module also supports the use of safe primes defined in RFC3526, which are part of the Modular Exponential (MODP) Diffie-Hellman groups that can be used for Internet Key Exchange (IKE). Note that the module only implements key generation and shared secret computation of safe primes, and no other part of the IKE protocol (with the exception of the IKE KDF, which is separately implemented).

- IKEv2 (RFC3526)
  - MODP-2048 (ID=14)
  - MODP-3072 (ID=15)
  - MODP-4096 (ID=16)
  - MODP-6144 (ID=17)
  - MODP-8192 (ID=18)

Additionally, the vendor also claims compliance to SP 800-56A rev3 key agreement an approved algorithm per IG D.8 scenario X1 path (2). Specifically, the module implements shared secret computation (KAS SSC) followed by the derivation of the keying material using SP 800-135 TLS KDF listed in IG G.20.

According to Table 2: Comparable strengths in [SP 800-57], the key sizes of AES, Triple-DES, RSA, Diffie-Hellman and EC Diffie-Hellman provide the following security strength in FIPS mode of operation:

- Diffie-Hellman shared secret computation provides between 112 and 200 bits of encryption strength.
- EC Diffie-Hellman shared secret computation provides between 128 and 256 bits of encryption strength.
- Diffie-Hellman key agreement with TLS KDF provides between 112 and 200 bits of encryption strength.
- EC Diffie-Hellman key agreement with TLS KDF provides between 128 and 256 bits of encryption strength.
- RSA key wrapping with PKCS#1-v1.5 provides between 112 and 256 bits of encryption strength considering the use of keys equal to or larger than 2048 bits up to 16384 bits; Allowed per IG D.9
- AES key wrapping with KW, KWP and GCM; KTS (AES Certs. #A1059, #A1072, #A1097 and #A1179; key establishment methodology provides between 128 and 256 bits of encryption strength).
- AES key wrapping with AES CBC and HMAC; KTS (AES Certs. #A1097 and #A1179 and HMAC Certs. #A1179; key establishment methodology provides 128 or 256 bits of encryption strength).
- Triple-DES Key wrapping with Triple-DES CBC and HMAC; KTS (Triple-DES Cert. #A1179 and HMAC Certs. #A1179; key establishment methodology provides 112 bits of encryption strength).

8.6 Key Derivation

The module supports the following key derivation methods:

- KDF for the TLS protocol, used as pseudo-random functions (PRF) for TLSv1.0/1.1 and TLSv1.2.
- HKDF for the TLS protocol TLSv1.3.
- KDF for the IKE protocol.

The module also supports password-based key derivation (PBKDF2). The implementation is compliant with option 1a of [SP 800-132]. Keys derived from input passwords or passphrases using this method can only be used in storage applications.

9. Self-Tests

FIPS 140-2 requires that the module perform self-tests to ensure the integrity of the module, and the correctness of the cryptographic functionality at start up. In addition, conditional tests are required during operational stage of the module. All of these tests are listed and described in this section.

9.1 Power-Up Self-Tests

All the power-up self-tests are performed automatically by initializing or re-initializing the module without requiring any operator intervention. During the power-up self-tests, no cryptographic operation is available and all input or output is inhibited. Once the power-up self-tests are completed successfully, the module enters operational mode and cryptographic operations are available. If any of the power-up self-tests fail, the module enters the Error state. In Error state, all output is inhibited and no cryptographic operation is allowed. The module returns the error code CKR_DEVICE_ERROR to the calling application to indicate the Error state. The module needs to be reinitialized in order to recover from the Error state. The following table provides the lists of Known-Answer Test (KAT) and Integrity Test as the power up self-tests:

Algorithm	Test
AES	KATs for ECB and CBC, modes: encryption and decryption are tested separately
AES-GCM	KAT for encryption and decryption
Triple-DES	KATs for ECB and CBC modes: encryption and decryption are tested separately
Diffie-Hellman	Primitive “Z” Computation KAT with 2048-bit key
DSA	KAT: signature generation and verification are tested separately
EC Diffie-Hellman	Primitive “Z” Computation KAT with P-256 curve
ECDSA	KAT: signature generation and verification are tested separately
RSA	KAT: encryption and decryption are tested separately KAT: signature generation and verification are tested separately
SHA	KAT: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
HMAC	KAT: HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512
CMAC	KAT: AES-CMAC KAT
PBKDF2 KDF	KAT: Using SHA-256
TLS KDF	KAT: TLS 1.0 PRF KAT and TLS 1.2 KAT using SHA-224, SHA-256, SHA-384 and SHA-512
HKDF KDF	KAT: HKDF KAT using SHA-256, SHA-384 and SHA-512
IKE KDF	KAT: SP800-135 IKE PRF using SHA-1, SHA-256, SHA-384 and SHA-512.
DRBG	KAT of Hash_DRBG with SHA-256
DRBG	DRBG health tests as specified in section 11.3 of NIST SP 800-90Ar1
Module Integrity	DSA signature verification with 2048 bits key and SHA-256

Table 11: Power-On Self-Tests

The power-up self-tests can be performed on demand by reinitializing the module.

9.2 Conditional Self-Tests

The following table provides the lists of Pairwise Consistency Test (PCT) and Continuous Random Number Generation 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 in order to recover from the Error state.

Algorithm	Test
DSA	PCT for DSA key generation
ECDSA	PCT for ECDSA key generation
RSA	PCT for RSA key generation
DRBG	CRNGT

Table 12: Conditional Self-Tests

10. Crypto-Officer and User Guidance

10.1 Crypto-Officer Guidance

The version of the RPM containing the validated module is stated in section 3.1 above. The RPM package of the Module shall be downloaded from the Oracle Linux 7 "Security Validation (Update 8)" yum repository and installed by standard tools recommended for the installation of Oracle packages on an Oracle Linux system (for example, yum, RPM, and the RHN remote management tool). The integrity of the RPM is automatically verified during the installation of the Module and the Crypto Officer shall not install the RPM file if the Oracle Linux Yum Server indicates an integrity error. The RPM files listed in section 3 are signed by Oracle and during installation; Yum performs signature verification which ensures as secure delivery of the cryptographic module. If the RPM packages are downloaded manually, then the CO should run 'rpm –K <rpm-file-name>' command after importing the builder's GPG key to verify the package signature. In addition, the CO shall also verify the hash of the RPM package to confirm a proper download.

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

To configure the operating environment to support FIPS Approved mode, shall perform the following steps:

1. Install nss-softokn RPM file e.g for x86_64 or aarch use yum command
   # yum install nss-softokn-3.53.1-6.0.1.el7_9.x86_64.rpm or nss-softokn-3.53.1-6.0.1.el7_9.aarch64.rpm
2. Install nss-softokn-freebl RPM file
   # yum install nss-softokn-freebl-3.53.1-6.0.1.el7_9.x86_64.rpm or nss-softokn-freebl-3.53.1-6.0.1.el7_9.aarch64.rpm
3. Install the dracut-fips package
   # yum install dracut-fips-033-572.0.5.el7.x86_64.rpm or dracut-fips-033-572.0.5.el7.aarch64.rpm
4. Install the dracut-fips-aesni package (if AES-NI or ARM optimizations are supported):
   To check if AES-NI is supported run
   # grep aes /proc/cpuinfo
   If it is supported, run:
   # yum install dracut-fips-aesni-033-572.0.5.el7.x86_64.rpm or dracut-fips-aesni-033-572.0.5.el7.aarch64.rpm
5. Recreate the INITRAMFS image:
   # dracut -f
6. Perform the following steps to configure the boot loader:

   a) Identify the boot partition and the UUID of the partition. If /boot or /boot/efi resides on a separate partition, the kernel parameter boot=<partition of /boot or /boot/efi> shall be supplied. The partition can be identified with the command:

      # df /boot or df /boot/efi

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

      # blkid /dev/sda1

/dev/sda1: UUID="6046308a-75fc-418e-b284-72d8bfad34ba" TYPE="xfs"

b) As the root user, edit the /etc/default/grub file as follows:

i. Add the fips=1 option to the boot loader configuration.

```
GRUB_CMDLINE_LINUX="vconsole.font=latarcyrheb-sun16
rd.lvm.lv=ol/swap rd.lvm.lv=ol/root crashkernel=auto
vconsole.keymap=uk rhgb quiet fips=1"
```

ii. If the contents of /boot reside on a different partition to the root partition, you must use the boot=UUID=boot_UUID line to the boot loader configuration to specify the device that should be mounted onto /boot when the kernel loads.

```
GRUB_CMDLINE_LINUX="vconsole.font=latarcyrheb-sun16
rd.lvm.lv=ol/swap rd.lvm.lv=ol/root crashkernel=auto
vconsole.keymap=uk rhgb quiet
boot=UUID=6046308a-75fc-418e-b284-72d8bfad34ba fips=1"
```

iii. Save the changes.

This is required for operating system kernel validation checks, where the kernel will be verified against the provided HMAC file in the /boot directory.

Note:
On systems that are configured to boot with UEFI, /boot/efi is located on a dedicated partition as this is formatted specifically to meet UEFI requirements. This does not automatically mean that /boot is located on a dedicated partition.

Only use the boot= parameter if /boot is located on a dedicated partition. If the parameter is specified incorrectly or points to a non-existent device, the system may not boot.

If the system is no longer able to boot, you can try to modify the kernel boot options in grub to specify an alternate device for the boot=UUID=boot_UUID parameter, or remove the parameter entirely.

7. Rebuild the GRUB configuration as follows:

On BIOS-based systems, run the following command:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

On UEFI-based systems, run the following command:

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

To ensure proper operation of the in-module integrity verification, prelinking shall be disabled on all system files. By default, the prelink package is not installed on the system. However, if it is installed, disable prelinking on all libraries and binaries as follows:

Set PRELINKING=no in the /etc/sysconfig/prelink configuration file.

If the libraries were already prelinked, undo the prelink on all of the system files as follows:

prelink –u –a

8. Reboot the system

9. Verify that FIPS Mode is enabled in the Operating System by running the command:

cat /proc/sys/crypto/fips_enabled

The response should be “1”

The version of the RPM containing the validated Modules is the version listed in Section 3. The integrity of the RPM is automatically verified during the installation of the Modules and the Crypto Officer shall not install the RPM file if the RPM tool indicates an integrity error.

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

10.1.1 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 shall be turned on as part of the initialization procedures by setting the environment variable NSS_ENABLE_AUDIT to 1. The Crypto-Officer shall 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 the /var/log directory. The exact location of the system log is specified in the /etc/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/lib64/libaudit.so.0 to /usr/lib64/libaudit.so.1.0.0 (on 64-bit platforms).

10.2 User Guidance

In order to run the module in FIPS-Approved mode, only the FIPS Approved or allowed services listed in Table 8 with the validated or allowed cryptographic algorithms/security functions listed in Table 2 and Table 3 shall be used.

The following module initialization steps shall be followed before starting to use the NSS module:

- Set the environment variable 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 shall be reset and initialized again.
- For the first login, provide a NULL user 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 initial NSS User password.
- Now, set the 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 NSS User role can now be assumed on the module by logging in using the User password. And the Crypto-Officer role can be implicitly assumed by performing the Crypto-Officer services as listed in Section 7.2.

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 user passwords and other security-relevant data items are maintained under the control of the cryptographic module. Secret and private keys shall be passed to the calling application in encrypted (wrapped) form with FC_WrapKey and entered from 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 user password. In such a case, they should be considered to be in plaintext form.

Automated key transport methods shall use FC_WrapKey and FC_UnwrapKey to output or input secret and private keys from or to the module.

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

10.2.1 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. It is the User’s responsible for ensuring the module’s compliance with this requirement

10.2.2 Key derivation using SP800-132 PBKDF2

The module provides password-based key derivation (PBKDF2), compliant with SP800-132. The module supports option 1a from section 5.4 of [SP800-132], in which the Master Key (MK) or a segment of it is used directly as the Data Protection Key (DPK).

In accordance with [SP800-132], the following requirements shall be met.

- Derived keys shall only be used in storage applications. The Master Key (MK) shall not be used for other purposes. The length of the MK or DPK shall be of 112 bits or more.

- A portion of the salt, with a length of at least 128 bits, shall be generated randomly using the SP800-90A DRBG,
- The iteration count shall be selected as large as possible, as long as the time required to generate the key using the password input to the PBKDF2, is acceptable for the users. The minimum value shall be 1000.
- Passwords or passphrases, used as an input for the PBKDF2, shall not be used as cryptographic keys.
- The length of the password or passphrase used as input to PBKKDF2 shall be of at least 20 characters, and shall consist of lower-case, upper-case and numeric characters. The probability of guessing the value is estimated to be $1/62^{20} = 10^{-36}$, which is less than 2^{-112}.

The calling application shall also observe the rest of the requirements and recommendations specified in [SP800-132].

10.2.3 AES-GCM IV

In case the module's power is lost and then restored, the key used for the AES GCM encryption or decryption shall be redistributed.

The nonce_explicit part of the IV does not exhaust the maximum number of possible values for a given session key. The design of the TLS protocol in this module implicitly ensures that the nonce_explicit, or counter portion of the IV will not exhaust all of its possible values.

The AES GCM IV generation is in compliance with the [RFC5288] and shall only be used for the TLS protocol version 1.2 to be compliant with [FIPS140-2_IG] IG A.5, provision 1 ("TLS protocol IV generation"); thus, the module is compliant with [SP800-52]. The module supports the TLS GCM ciphersuites from SP800-52 Rev1, section 3.3.1

When a GCM IV is used for decryption, the responsibility for the IV generation lies with the party that performs the AES GCM encryption.

10.3 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.

11. Mitigation of Other Attacks

The module is designed to mitigate the following attacks.

Attack	Mitigation Mechanism	Specific Limit
Timing attacks on RSA	**RSA blinding** Timing attack on RSA was first demonstrated by Paul Kocher in 1996 [15], who contributed the mitigation code to our module. Most recently Boneh and Brumley [16] 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 modular exponentiation** This is a variant of a modular exponentiation implementation that Colin Percival [17] 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.
Arithmetic errors in RSA signatures	**Double-checking RSA signatures** Arithmetic errors in RSA signatures might leak the private key. Ferguson and Schneier [18] recommend that every RSA signature generation should verify the signature just generated.	None.

Table 13: Mitigation of Other Attacks

Acronyms, Terms and Abbreviations

Term	Definition
AES	Advanced Encryption Standard
AES-NI	Intel Advanced Encryption Standard New Instructions
CBC	Cipher Block Chaining
CMVP	Cryptographic Module Validation Program
CCCS	Canadian Centre for Cyber Security
CMAC	Cipher-Based Message Authentication Code
CSP	Critical Security Parameter
CTR	Counter Block Chaining
CTS	Ciphertext Stealing
CVL	Component Validation List
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
ECDSA	Elliptic Curve Digital Signature Algorithm
DRBG	Deterministic Random Bit Generator
GCM	Galois/Counter Mode
HMAC	(Keyed) Hash Message Authentication Code
IKE	Internet Key Exchange
MAC	Message Authentication Code
KAT	Known Answer Test
KDF	Key Derivation Function
NIST	National Institute of Standards and Technology
NSS	Network Security Services
PBKDF2	Password Based Key Derivation Function
PKCS	Public-Key Cryptographic Standard
PUB	Publication
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
TLS	Transport Layer Security

Table 14: Acronyms

References

The FIPS 140-2 standard, and information on the CMVP, can be found at http://csrc.nist.gov/groups/STM/cmvp/index.html. More information describing the module can be found on the Oracle web site at https://www.oracle.com/linux/ .

This Security Policy contains non-proprietary information. All other documentation submitted for FIPS 140-2 conformance testing and validation is "Oracle - Proprietary" and is releasable only under appropriate non-disclosure agreements.

[1] FIPS 140-2 Standard, http://csrc.nist.gov/groups/STM/cmvp/standards.html
[2] FIPS 140-2 Implementation Guidance, https://csrc.nist.gov/CSRC/media/Projects/cryptographic-module-validation-program/documents/fips140-2/FIPS1402IG.pdf
[3] FIPS 140-2 Derived Test Requirements, https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Module-Validation-Program/documents/fips140-2/FIPS1402DTR.pdf
[4] FIPS 197 Advanced Encryption Standard, https://csrc.nist.gov/publications/detail/fips/197/final
[5] FIPS 180-4 Secure Hash Standard, https://csrc.nist.gov/publications/detail/fips/180/4/final
[6] FIPS 198-1 The Keyed-Hash Message Authentication Code (HMAC), https://csrc.nist.gov/publications/detail/fips/198/1/final
[7] FIPS 186-4 Digital Signature Standard (DSS), https://csrc.nist.gov/publications/detail/fips/186/4/final
[8] NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation: Methods and Techniques, https://csrc.nist.gov/publications/detail/sp/800-38a/final
[9] NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, https://csrc.nist.gov/publications/detail/sp/800-38d/final
[10] NIST SP 800-38F, Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping, https://csrc.nist.gov/publications/detail/sp/800-38f/final
[11] NIST SP 800-56A Revision 3, Recommendation for Pair-Wise Key Establishment Schemes using Discrete Logarithm Cryptography (Revised), https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final
[12] NIST SP 800-67 Revision 2, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, https://csrc.nist.gov/publications/detail/sp/800-67/rev-2/final
[13] NIST SP 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final
[14] RSA Laboratories, "PKCS #11 v2.20: Cryptographic Token Interface Standard", 2004.
[15] 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/
[16] D. Boneh and D. Brumley, "Remote Timing Attacks are Practical", http://crypto.stanford.edu/~dabo/abstracts/ssl-timing.html
[17] C. Percival, "Cache Missing for Fun and Profit", http://www.daemonology.net/papers/htt.pdf
[18] N. Ferguson and B. Schneier, Practical Cryptography, Sec. 16.1.4 "Checking RSA Signatures", p. 286, Wiley Publishing, Inc., 2003.