Red Hat Enterprise Linux 8 NSS Cryptographic Module

version rhel8.20190808

FIPS 140-2 Non-Proprietary Security Policy

Document Version 1.1

Last Update: 2021-01-13

Table of Contents

1. Cryptographic Module Specification

This document is the non-proprietary security policy for the Red Hat Enterprise Linux 8 NSS Cryptographic Module, and was prepared as part of the requirements for conformance to Federal Information Processing Standard (FIPS) 140-2, Security Level 1.

1.1. Description of the Module

The Red Hat Enterprise Linux 8 NSS Cryptographic Module version rhel8.20190808 (hereafter referred to as the “Module”) is a software library supporting FIPS 140-2 approved cryptographic algorithms. For the purposes of the FIPS 140-2 validation, its embodiment type is defined as multi-chip standalone. 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 FIPS 140-2 validated at overall Security Level 1 with levels for individual sections shown in the table below:

Security Component	FIPS 140-2 Security Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services and Authentication	2
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	2
Mitigation of Other Attacks	1

Table 1: Security Level of the Module

The Red Hat Enterprise Linux 8 NSS Cryptographic Module has been tested on the following platforms:

Hardware Platform	Processor	Operating System	Tested	
			With AES-NI	Without AES-NI
Dell PowerEdge R430	Intel(R) Xeon(R) E5	Red Hat Enterprise Linux 8	Yes	Yes

Table 2: Tested Platforms

1.2. Description of the Approved Modes

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.

The table below lists the Approved algorithms in FIPS Approved mode:

Usage	Approved Algorithm	Keys/CSPs	CAVS Certificate
Encryption and decryption	AES encryption and decryption with ECB, CBC and CTR modes	AES 128, 192 and 256 bits keys	Certs. #A132, #A136
	Three-key Triple-DES encryption and decryption with ECB and CBC modes	Three-key Triple-DES 168 bits keys	Cert. #A132
Message digest	SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512	N/A	Cert. #A132
	HMAC with SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512	At least 112 bits HMAC keys	Cert. #A132
Random number generation	NIST SP800-90A Hash_DRBG with SHA-256	Entropy input string, seed, V and C values	Cert. #A132
Signature generation and verification	DSA signature generation	L=3072, N=256 (with SHA-224, SHA-256 L=2048, N=256 (with SHA-224, SHA-256 L=2048, N=224 (with SHA-224, SHA-256	Cert. #A132
	DSA signature verification	L=3072, N=256 (with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) L=2048, N=256 (with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) L=2048, N=224 (with SSHA-1, SHA-224, SHA-256, SHA-384, SHA-512) L=1024, N=160 (with SSHA-1, SHA-224, SHA-256, SHA-384, SHA-512)	

Usage	Approved Algorithm	Keys/CSPs	CAVS Certificate
	ECDSA signature generation	ECDSA keys based on P-256, P-384 and P-521 curves (with SHA-224, SHA-256, SHA-384, SHA-512)	
	ECDSA signature verification	ECDSA keys based on P-256, P-384 and P-521 curves (with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512)	
	RSA signature generation according to PKCS#1 v1.5	RSA 2048, 3072 and 4096 bits keys (with SHA-256, SHA-384, SHA-512)	
	RSA signature verification according to PKCS#1 v1.5	RSA 2048, 3072 and 4096 bits keys (with SHA-1, SHA-256, SHA-384, SHA-512)	
Key and domain parameter generation	RSA key pair generation	RSA 2048, 3072 and 4096 bits keys	Cert. #A132
	DSA key pair generation	DSA 2048 and 3072 bits keys	
	DSA domain parameter generation	L=3072, N=256 (with SHA-256 L=2048, N=256 (with SHA-256 L=2048, N=224 (with SHA-224	
	ECDSA key pair generation	ECDSA keys based on P-256, P-384 and P-521 curves	
Key and domain parameter verification	ECDSA public key verification	ECDSA keys based on P-256, P-384 and P-521 curves	
KAS FCC	Shared Secret Computation	DH 2048-bit key with SHA-224 and SHA-256	CVL Cert. #A132
KAS ECC		ECDH key based on P-256, P-384 and P-521	
Key derivation	SP800-135 key derivation (TLS v1.0, TLS v1.1 and TLS v1.2)	TLS pre-master secret and master secret	CVL Cert. #A132

Usage	Approved Algorithm	Keys/CSPs	CAVS Certificate
	SP800-135 key derivation (IKE v1 and v2)	IKE SA encryption keys and integrity keys, IPsec SA encryption keys and integrity keys, shared keys and shared secret	CVL Cert. #A173
	SP800-132 PBKDF	PBKDF password PBKDF derived key	(vendor affirmed)[1]
Key generation	SP800-133 Cryptographic Key Generation (CKG)	AES key Triple-DES key HMAC key RSA key pair DSA key pair ECDSA key pair ECDH key pair DH key pair	(vendor affirmed)

Table 3: Approved Algorithms in FIPS Approved mode

Note: The TLS protocol has not been reviewed or tested by the CAVP and CMVP.

Note: There are algorithms, modes, and keys that have been CAVP tested but not used by the module in FIPS approved mode. Only the algorithms, modes/methods, and key lengths/curves/moduli shown in tables 3 and 4 are used by the module in FIPS approved mode.

Table 4 lists the non-Approved but allowed algorithms in FIPS Approved mode:

Usage	non-Approved but allowed Algorithm	Keys/CSPs	Note
Key wrapping	RSA key wrapping (encrypt, decrypt)	RSA keys (>= 2048 up to 15360 bits or more)	Not compliant with NIST SP 800-56B, but allowed in FIPS mode t according to IG D.9
Key agreement	Diffie-Hellman key agreement	Diffie-Hellman public and private components with size between 2048 bits and 15360 bits	Allowed in FIPS mode according to IG D.8 CVL Cert. #A132
	EC Diffie-Hellman key agreement	EC Diffie-Hellman public and private components based on P-256, P-384 and P-521 curves	Allowed in FIPS mode according to IG D.8 CVL Cert. #A132
Random number generation	NDRNG	N/A	getrandom() is used to seed the module's SP 800-

1The vendor claims compliance to the SP 800-132 PBKDF2, which was tested through ACVP obtaining Cert. #A132. However, the module does not implement a KAT for this algorithm and hence it is claimed as vendor affirmed.

Usage	non-Approved but allowed Algorithm	Keys/CSPs	Note
			90A DRBG

Table 4: non-Approved but Allowed Algorithms in FIPS Approved mode

Notes:

1. RSA (key wrapping; key establishment methodology provides between 112 and 256 bits of encryption strength; non-compliant less than 112 bits of encryption strength)
2. Diffie-Hellman (key agreement; key establishment methodology provides between 112 and 256 bits of encryption strength; non-compliant less than 112 bits of encryption strength)
3. EC Diffie-Hellman (key agreement; key establishment methodology provides between 128 and 256 bits of encryption strength)

Table 5 lists the non-Approved algorithms, which invocation will result the Module operating in non-Approved mode implicitly.

Usage	non-Approved Algorithm
Encryption and decryption	AES CTS and GCM mode
	Camellia
	Chacha20/Poly1305 AEAD
	DES
	RC2
	RC4
	RC5
	SEED
	Two-key Triple-DES encryption/decryption
Signature generation and verification	DSA signature generation with key size not equal to 2048 or 3072 bits DSA signature generation using SHA-1, SHA-384 and SHA-512 DSA signature verification with key size not equal to 1024, 2048 or 3072 bits
	RSA signature generation with key size not equal to 2048 or 3072 bits RSA signature generation using SHA-1 RSA signature generation and signature verification using SHA-224 RSA signature verification with key size not equal to 1024, 2048 or 3072 bits
	ECDSA signature generation using SHA-1
Key Derivation	PBKDF (non-complient with SP800-132)
Message digest	MD2
	MD5
Key management	DSA domain parameter verification with key size not equal to 1024, 2048 or 3072 bits DSA key pair generation with key size not equal to 2048 and 3072 bits

Usage	non-Approved Algorithm
	RSA key pair generation for key sizes not listed in Table 3
	AES/Triple-DES non-SP 800-38F compliant key wrapping
	Diffie-Hellman key agreement with key size less than 2048 bits
	RSA key wrapping (encrypt, decrypt) with key size less than 2048 bits
	J-PAKE key agreement

Table 5: non-Approved Algorithms

1.3. Cryptographic Boundary

The Module's physical boundary is the surface of the case of the platform (depicted in Figure 1).

The Module's logical cryptographic boundary consists of the shared library files and their integrity check signature files, which are delivered through Red Hat Package Manager (RPM) as listed below:

- nss-softokn RPM file with version 3.39.0-1.3.el8, which contains the following files:
  - /usr/lib64/libsoftokn3.chk (64 bits)
  - /usr/lib64/libsoftokn3.so (64 bits)
- nss-softokn-freebl RPM with version 3.39.0-1.3.el8, which contains the following files:
  - /usr/lib64/libfreeblpriv3.chk (64 bits)
  - /usr/lib64/libfreeblpriv3.so (64 bits)

◦

1.3.1. Hardware Block Diagram

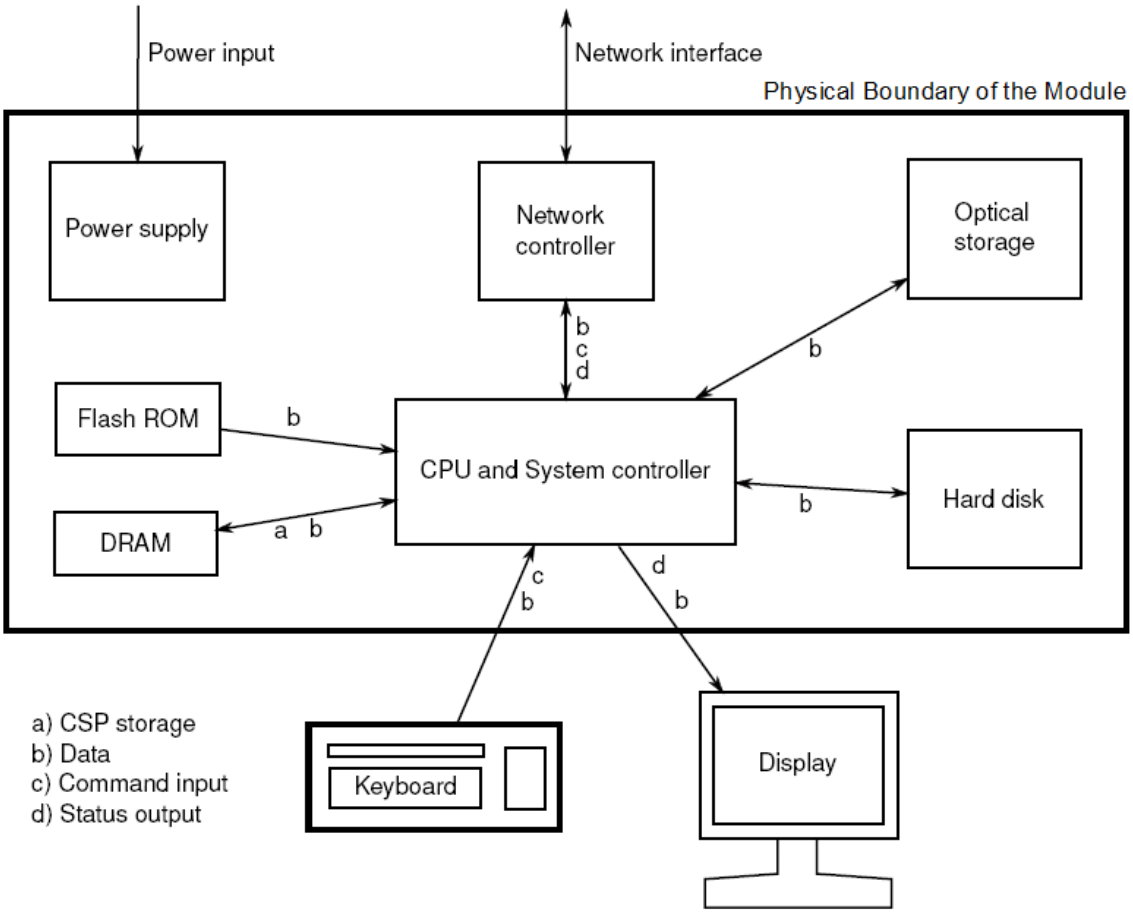

Figure 1: Hardware Block Diagram

1.3.2. Software Block Diagram

The NSS cryptographic module implements the PKCS #11 (Cryptoki) API. The API itself defines the logical cryptographic boundary, thus all implementation is inside the boundary. The diagram below shows the relationship of the layers.

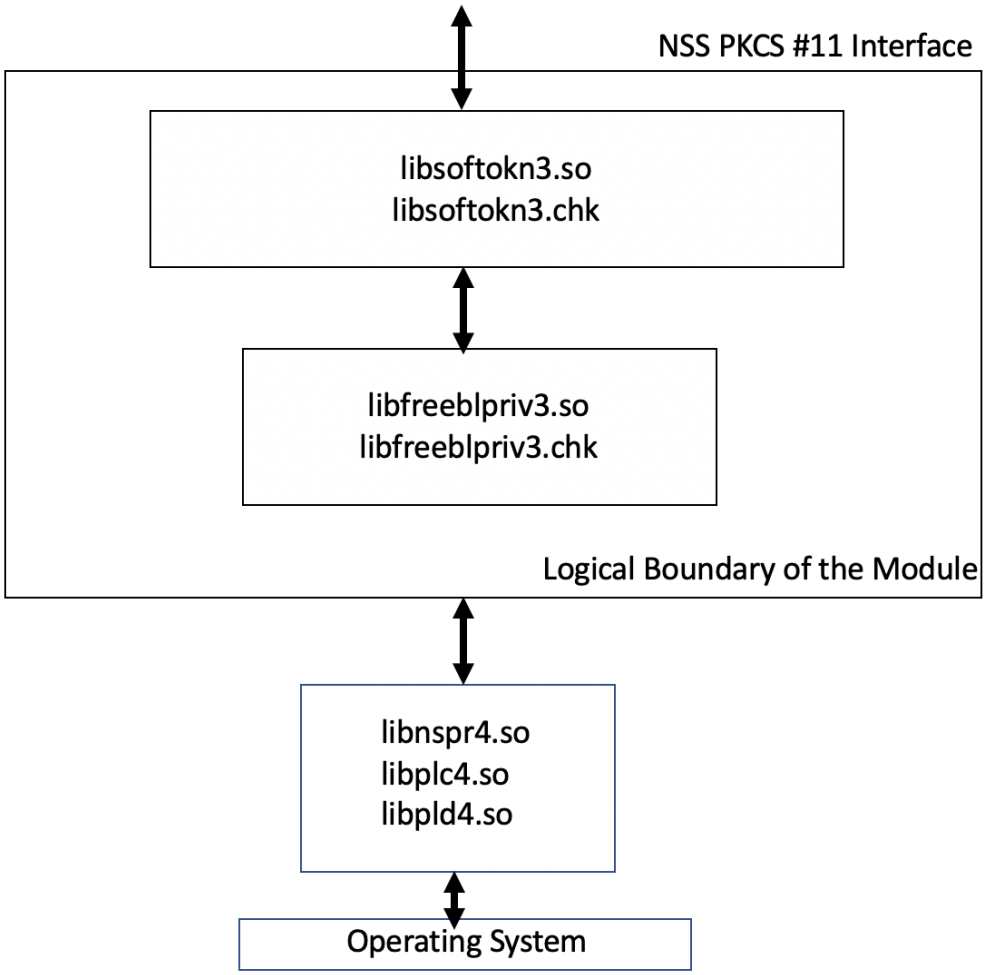

Figure 2: Software Block Diagram

2. Cryptographic Module Ports and Interfaces

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, the environment variables and configuration file.

Table 6 Summarizes the four logical interfaces.

FIPS 140-2 Interface	Logical Interface
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, environment variables and configuration file (/proc/sys/crypto/fips_enabled)
Status Output	API return codes and status parameters

Table 6: Ports and Interfaces

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.

2.1. 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 7.

2.2. 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`.

2.3. 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.

3. Roles, Services and Authentication

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

3.1. 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.

3.2. Role Assumption

The CO role is implicitly assumed by an operator while installing the Module by following the instructions in Section 9.1 and while performing other CO services on the Module.

The Module implements a password-based authentication for the User role (role-based authentication). 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 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 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 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 the Module is powered off, the result of previous authentication will be cleared and the user needs to be re-authenticated.

3.3. 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: digits (0-9), ASCII lowercase letters (a-z), ASCII uppercase letters (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 '¿'). 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.

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

- The characters of the password are independent with 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 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 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.

3.4. Multiple Concurrent Operators

The Module doesn't allow concurrent operators.

Note: FIPS 140-2 Implementation Guidance Section 6.1 clarifies the use of a cryptographic module on a server.

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

3.5. Services

3.5.1. Calling Convention of API Functions

The Module has a set of API functions denoted by FC_xxx. All the API functions are listed in Table 7.

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 examples:

- 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.

3.5.2. API Functions

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 7 lists all the services available in FIPS Approved mode with the role type, API function, description, Keys/CSPs and access type. Access types R, W and Z stand for Read, Write, and Zeroize, respectively. Role types U and CO correspond to User role and Crypto Officer role, respectively. Please refer to Table 3and Table 4 for the Approved or allowed cryptographic algorithms supported by the Module.

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.

Service	Role	Function	Description	Keys/CSPs	Access
Get the function list	CO	FC_GetFunctionList	Return a pointer to the list of function pointers for the operational mode	none	-
Module initialization	CO	FC_InitToken	Initialize or re-initialize a token	User password and all keys	Z
	CO	FC_InitPIN	Initialize the user's password, i.e., set the user's initial password	User password	W
General Purpose	CO	FC_Initialize	Initialize the module library	none	-
	CO	FC_Finalize	Finalize (shut down) the module library	All keys	Z
	CO	FC_GetInfo	Obtain general information about the module library	none	-
Slot and token management	CO	FC_GetSlotList	Obtain a list of slots in the system	none	-
	CO	FC_GetSlotInfo	Obtain information about a particular slot	none	-
	CO	FC_GetTokenInfo	Obtain information about the token (This function provides the Show Status service)	none	-
	CO	FC_GetMechanismList	Obtain a list of mechanisms (cryptographic algorithms) supported by a token	none	-
	CO	FC_GetMechanismInfo	Obtain information about a particular mechanism	none	-

Service	Role	Function	Description	Keys/CSPs	Access
	U	FC_SetPIN	Change the user's password	User password	RW
Session management	CO	FC_OpenSession	Open a connection (session) between an application and a particular token	none	-
	CO	FC_CloseSession	Close a session	All keys for the session	Z
	CO	FC_CloseAllSessions	Close all sessions with a token	All keys	Z
	CO	FC_GetSessionInfo	Obtain information about the session (This function provides the Show Status service)	none	-
	CO	FC_GetOperationState	Save the state of the cryptographic operations in a session (This function is only implemented for message digest operations)	none	-
	CO	FC_SetOperationState	Restore the state of the cryptographic operations in a session (This function is only implemented for message digest operations)	none	-
	U	FC_Login	Log into a token	User password	R
	U	FC_Logout	Log out from a token	none	-
Object management	U	FC_CreateObject	Create a new object	key	W
	U	FC_CopyObject	Create a copy of an object	Original key	R
				New key	W
	U	FC_DestroyObject	Destroy an object	key	Z
	U	FC_GetObjectSize	Obtain the size of an object in bytes	key	R
	U	FC_GetAttributeValue	Obtain an attribute value of an object	key	R
	U	FC_SetAttributeValue	Modify an attribute value of an object	key	W
	U	FC_FindObjectsInit	Initialize an object search operation	none	-

Service	Role	Function	Description	Keys/CSPs	Access
	U	FC_FindObjects	Continue an object search operation	Keys matching the search criteria	R
	U	FC_FindObjectsFinal	Finish an object search operation	none	-
Encryption and decryption	U	FC_EncryptInit	Initialize an encryption operation	AES/Triple-DES key	R
	U	FC_Encrypt	Encrypt single-part data	AES/Triple-DES key	R
	U	FC_EncryptUpdate	Continue a multiple-part encryption operation	AES/Triple-DES key	R
	U	FC_EncryptFinal	Finish a multiple-part encryption operation	AES/Triple-DES key	R
	U	FC_DecryptInit	Initialize a decryption operation	AES/Triple-DES key	R
	U	FC_Decrypt	Decrypt single-part encrypted data	AES/Triple-DES key	R
	U	FC_DecryptUpdate	Continue a multiple-part decryption operation	AES/Triple-DES key	R
	U	FC_DecryptFinal	Finish a multiple-part decryption operation	AES/Triple-DES key	R
Message digest	CO	FC_DigestInit	Initialize a message-digesting operation	none	-
	CO	FC_Digest	Digest single-part data	none	-
	CO	FC_DigestUpdate	Continue a multiple-part digesting operation	none	-
	U	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
	CO	FC_DigestFinal	Finish a multiple-part digesting operation	none	-
Signature generation and verification	U	FC_SignInit	Initialize a signature operation	DSA/ECDSA/RSA private key, HMAC key	R
	U	FC_Sign	Sign single-part data	DSA/ECDSA/RSA private key, HMAC key	R
	U	FC_SignUpdate	Continue a multiple-part signature operation	DSA/ECDSA/RSA private key,	R

Service	Role	Function	Description	Keys/CSPs	Access
				HMAC key	
	U	FC_SignFinal	Finish a multiple-part signature operation	DSA/ECDSA/RSA private key, HMAC key	R
	U	FC_SignRecoverInit	Initialize a signature operation, where the data can be recovered from the signature	DSA/ECDSA/RSA private key	R
	U	FC_SignRecover	Sign single-part data, where the data can be recovered from the signature	DSA/ECDSA/RSA private key	R
	U	FC_VerifyInit	Initialize a verification operation	DSA/ECDSA/RSA public key, HMAC key	R
	U	FC_Verify	Verify a signature on single-part data	DSA/ECDSA/RSA public key, HMAC key	R
	U	FC_VerifyUpdate	Continue a multiple-part verification operation	DSA/ECDSA/RSA public key, HMAC key	R
	U	FC_VerifyFinal	Finish a multiple-part verification operation	DSA/ECDSA/RSA public key, HMAC key	R
	U	FC_VerifyRecoverInit	Initialize a verification operation, where the data is recovered from the signature	DSA/ECDSA/RSA public key	R
	U	FC_VerifyRecover	Verify a signature on single-part data, where the data is recovered from the signature	DSA/ECDSA/RSA public key	R
Dual-function cryptographic operations	U	FC_DigestEncryptUpdate	Continue a multiple-part digesting and encryption operation	AES/Triple-DES key	R
	U	FC_DecryptDigestUpdate	Continue a multiple-part decryption and digesting operation	AES/Triple-DES key	R
	U	FC_SignEncryptUpdate	Continue a multiple-part signing and encryption operation	DSA/ECDSA/RSA private key, HMAC key	R
				AES/Triple-DES key	R
	U	FC_DecryptVerifyUpda	Continue a multiple-part	DSA/ECDSA/RSA	R

Service	Role	Function	Description	Keys/CSPs	Access
		te	decryption and verify operation	public key, HMAC key	
				AES/Triple-DES key	R
Key management	U	FC_GenerateKey	Generate a secret key (Also used by TLS to generate a pre-master secret) Used to derive a key from PBKDF password	AES/Triple-DES/HMAC key, TLS pre-master secret	W
				PBKDF derived key	
	U	FC_GenerateKeyPair	Generate a public/private key pair (This function performs the pair-wise consistency tests)	RSA key,DSA/ECDSA key pair, Diffie-Hellman/EC Diffie-Hellman public and private components	W
	U	FC_WrapKey	Wrap (encrypt) a key using the following mechanism: RSA encryption	Wrapping key	R
				Key to be wrapped	R
	U	FC_UnwrapKey	Unwrap (decrypt) a key using the following mechanism: RSA decryption	Unwrapping key	R
				Unwrapped key	W
	U	FC_DeriveKey	Derive a key from TLS master secret which is derived from TLS pre-master secret Derive a key from IKE shared secret	TLS pre-master secret	R
				TLS master secret	RW
				IKE shared secret	RW
				Derived key	W
Random number generation	CO	FC_SeedRandom	Mix in additional seed material to the random number generator	Entropy input string, seed, DRBG V and C values	RW
	CO	FC_GenerateRandom	Generate random data (This function performs the continuous random number generator test)	Random data, DRBG V and C values	RW
Parallel function management	CO	FC_GetFunctionStatus	A legacy function, which simply returns the value 0x00000051 (function	none	-

Service	Role	Function	Description	Keys/CSPs	Access
			not parallel)		
	CO	FC_CancelFunction	A legacy function, which simply returns the value 0x00000051 (function not parallel)	none	-
Self tests	CO	N/A	The self tests are performed automatically when loading the module	DSA 2048-bit public key for module integrity test	R
Show Status	U	N/A	Via exit codes	N/A	N/A
Zeroization	U	FC_DestroyObject	All CSPs are automatically zeroized when freeing the cipher handle	All secret or private keys and password	Z
	CO	FC_InitToken FC_Finalize FC_CloseSession FC_CloseAllSessions			

Table 7: Services details in FIPS Approved mode

Note:

1. *'Original key' and 'New key' are the secret keys or public/private key pairs.*
2. *'Wrapping key' corresponds to the secret key or public key used to wrap another key*
3. *'Key to be wrapped' is the key that is wrapped by the 'wrapping key'*
4. *'Unwrapping key' corresponds to the secret key or private key used to unwrap another key*
5. *'Unwrapped key' is the plaintext key that has not been wrapped by a 'wrapping key'*
6. *'Derived key' is the key obtained by a key derivation function which takes the 'TLS master secret' as input*

Table 7(A) 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 7, but the underneath non-Approved algorithms are invoked. Please also refer to Table 5 for the non-Approved algorithms. If any service invokes the non-Approved algorithms, then the module will enter non-Approved mode implicitly.

Service	Function	non-Approved Algorithm invoked
Encryption and decryption	FC_EncryptInit	AES CTS mode, AES-GCM, Camellia, ChaCha20/Poly1305 AEAD, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
	FC_Encrypt	
	FC_EncryptUpdate	
	FC_EncryptFinal	
	FC_DecryptInit	AES CTS mode, AES-GCM, Camellia, ChaCha20/Poly1305 AEAD, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
	FC_Decrypt	
	FC_DecryptUpdate	
	FC_DecryptFinal	

Service	Function	non-Approved Algorithm invoked
Message digest	FC_DigestInit	MD2, MD5
	FC_Digest	
	FC_DigestUpdate	
	FC_DigestKey	
	FC_DigestFinal	
Signature generation and verification	FC_SignInit	DSA signature generation with non-compliant key size listed in Table 5, RSA signature generation with non-compliant key size listed in Table 5 , RSA signature generation with SHA-1 and SHA-224 , ECDSA signature generation with SHA-1, DSA signature generation with SHA-1, SHA-384 and SHA-512
	FC_Sign	
	FC_SignUpdate	
	FC_SignFinal	
	FC_SignRecoverInit	
	FC_SignRecover	
	FC_VerifyInit	DSA signature verification with non-compliant key size listed in Table 5, RSA signature verification with non-compliant key size listed in Table 5
	FC_Verify	
	FC_VerifyUpdate	
	FC_VerifyFinal	
	FC_VerifyRecoverInit	
	FC_VerifyRecover	
Dual-function cryptographic operations	FC_DigestEncryptUpdate	MD2, MD5, AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
	FC_DecryptDigestUpdate	AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, MD2, MD5
	FC_SignEncryptUpdate	DSA signature generation with non-compliant key size listed in Table 5, RSA signature generation with non-compliant key size listed in Table 5, AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
	FC_DecryptVerifyUpdate	AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, DSA signature verification with non-compliant key size listed in Table 5, RSA signature verification with non-compliant key size listed in Table 5
Key management	FC_GenerateKeyPair	DSA domain parameter verification with non-compliant key size listed in Table 5, DSA key pair generation with non-compliant key size listed in Table 5, RSA key pair generation
	FC_WrapKey	AES key wrapping (encrypt) based on NIST SP800-38F, Triple-DES key wrapping (encrypt) using Two-key Triple-DES, RSA key wrapping (encrypt) with non-compliant key size listed in Table 5

Service	Function	non-Approved Algorithm invoked
	FC_UnwrapKey	AES key wrapping (decrypt) based on NIST SP800-38F, Triple-DES key wrapping (decrypt) using Two-key Triple-DES, RSA key wrapping (decrypt) with non-compliant key size listed in Table 5
	FC_DeriveKey	Diffie-Hellman key agreement with non-compliant key size listed in Table 5, J-PAKE key agreement, PBKDF(non-complient with SP800-132)

Table 7(A): Services details in non-Approved mode

4. Physical Security

The Module comprises of software only and thus does not claim any physical security.

5. Operational Environment

5.1 Applicability

The module operates in a modifiable operational environment per FIPS 140-2 level 1 specifications. The module runs on a commercially available general-purpose operating system executing on the hardware specified in section 2.2.

The Red Hat Enterprise Linux operating system is used as the basis of other products which include but are not limited to:

- Red Hat Enterprise Linux CoreOS
- Red Hat Virtualization (RHV)
- Red Hat OpenStack Platform
- OpenShift Container Platform
- Red Hat Gluster Storage
- Red Hat Ceph Storage
- Red Hat CloudForms
- Red Hat Satellite.

- Compliance is maintained for these products whenever the binary is found unchanged.

5.2 Policy

The operating system is restricted to a single operator (concurrent operators are explicitly excluded). The application that request cryptographic services is the single user of the module, even when the application is serving multiple clients.

In FIPS Approved mode, the ptrace(2) system call, the debugger (gdb(1)), and strace(1) shall be not used.

6. Cryptographic Key Management

The following table provides a summary of the Keys/CSPs in the Module:

Keys/CSPs	Generation	Storage	Entry/Output	Zeroization
AES 128, 192 and 256 bits keys	Use of NIST SP800-90A DRBG	Application memory or key database	Encrypted through key wrapping using `FC_UnwrapKey` for input and `FC_WrapKey` for output	Automatically zeroized when freeing the cipher handle
Triple-DES 168 bits keys	Use of NIST SP800-90A DRBG	Application memory or key database	Encrypted through key wrapping using `FC_UnwrapKey` for input and `FC_WrapKey` for output	Automatically zeroized when freeing the cipher handle
DSA 2048 and 3072 bits private keys	Use of NIST SP800-90A DRBG as a seed for the FIPS 186-4 DSA key generation mechanism	Application memory or key database	Encrypted through key wrapping using `FC_UnwrapKey` for input and `FC_WrapKey` for output	Automatically zeroized when freeing the cipher handle
ECDSA private keys based on P-256, P-384 and P-521 curves	Use of NIST SP800-90A DRBG as a seed for the FIPS 186-4 ECDSA key generation mechanism	Application memory or key database	Encrypted through key wrapping using `FC_UnwrapKey` for input and `FC_WrapKey` for output	Automatically zeroized when freeing the cipher handle
RSA 2048, 3072 and 4096 bits private keys	Use of NIST SP800-90A DRBG as a seed for the FIPS 186-4 RSA key generation mechanism	Application memory or key database	Encrypted through key wrapping using `FC_UnwrapKey` for input and `FC_WrapKey` for output	Automatically zeroized when freeing the cipher handle
HMAC keys with at least 112 bits	Use of NIST SP800-90A DRBG	Application memory or key data base	Encrypted through key wrapping using	Automatically zeroized when freeing the cipher

			FC_UnwrapKey for input and FC_WrapKey for output	handle
DRBG entropy input string and seed	Obtained from getrandom()	Application memory	N/A	Automatically zeroized when freeing DRBG handle
DRBG V and C values	Derived from the entropy input string as defined in NIST SP800-90A	Application memory	N/A	Automatically zeroized when freeing DRBG handle
TLS pre-master secret	Use of NIST SP800-90A DRBG in Diffie-Hellman or EC Diffie-Hellman key agreement scheme	Application memory	Encrypted for output using FC_WrapKey	Automatically zeroized when freeing the cipher handle
TLS master secret	Derived from TLS pre-master secret by using key derivation	Application memory	Encrypted for output using FC_WrapKey	Automatically zeroized when freeing the cipher handle
Diffie-Hellman private components with size between 2048 bits and 15360 bits	Use of NIST SP800-90A DRBG as a seed for the 186-4 DSA key generation mechanism	Application memory	N/A	Automatically zeroized when freeing the cipher handle
EC Diffie-Hellman private components based on P-256, P-384 and P-521 curves	Use of NIST SP800-90A DRBG as a seed for the 186-4 ECDSA generation mechanism	Application memory	N/A	Automatically zeroized when freeing the cipher handle
Shared secret according to the IKE protocol	Use of NIST SP800-90A DRBG as a seed for the SP800-56A (EC) Diffie-Hellman key agreement scheme shared-secret computation	Ephemeral	Encrypted for output using FC_WrapKey	Close of IKE SA or termination of Pluto IKE Daemon zeroizes the CSP
IKE SA Tunnel Encryption Keys	SP 800-135 IKE KDF	Ephemeral	Encrypted for output using FC_WrapKey	Close of IKE SA or termination of Pluto IKE Daemon

				zeroizes the CSP
IKE SA Tunnel Integrity Keys (HMAC keys)	SP 800-135 IKE KDF	Ephemeral	Encrypted for output using FC_WrapKey	Close of IKE SA or termination of Pluto IKE Daemon zeroizes the CSP
IPsec SA Tunnel Encryption Keys (AES or Triple-DES keys)	SP 800-135 IKE KDF	Ephemeral	Encrypted for output using FC_WrapKey	Zeroized from the module's memory when passed to the kernel after establishment of the SA
PBKDF password	N/A	Application memory	API input parameter	Automatically zeroized when freeing the cipher handle
PBKDF derived key	Derived using SP800-132 PBKDF mechanisms	Application memory	Encrypted for output using FC_WrapKey	Automatically zeroized when freeing the cipher handle
User Passwords	N/A (supplied by the calling application)	Application memory or key database in salted form	API input parameter	Automatically zeroized when the module is re-initialized or overwritten when the user changes its password

Table 8: Keys/CSPs

Note: The getrandom() is a function to access an NDRNG located within the module's physical boundary but outside the logical boundary.

6.1. Random Number Generation

The Module employs a NIST SP800-90 Hash_DRBG with SHA-256 as random number generator. The random number generator is seeded by obtaining random data from the operating system via getrandom(). The entropy source getrandom() provides at least 128 bits of random data available to the Module to obtain.

Reseeding is performed by pulling more data from getrandom(). 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 the DRBG health testing as specified in section 11.3 of NIST SP800-90A.

The Key Generation methods implemented in the module for Approved services in FIPS mode is compliant with [SP800-133].

For generating RSA, DSA and ECDSA keys the module implements asymmetric key generation services compliant with [FIPS186-4]. A seed (i.e. the random value) used in asymmetric key

generation is directly obtained from the [SP800-90A] DRBG.

The public and private key pairs used in the Diffie-Hellman and EC Diffie-Hellman KAS are generated internally by the module using the same DSA and ECDSA key generation compliant with [FIPS186-4] which is compliant with [SP800-56A].

The module generates symmetric key through the FC_GenerateKey() function using the random numbers from the SP 800-90A DRBG.

Caveat:

> The module generates cryptographic keys whose strengths are modified by available entropy.

6.2. Key/CSP Storage

The Module employs the cryptographic keys and CSPs in the FIPS Approved mode of operation as listed in Table 8. 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.

6.3. 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 password. A plaintext secret or private key gets 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_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 password.

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

7. Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

MARKETING NAME..... PowerEdge R430
REGULATORY MODEL..... E28S
REGULATORY TYPE..... E28S001
EFFECTIVE DATE..... December 02, 2014
EMC EMISSIONS CLASS..... Class A

7.1 Statement of compliance

This product has been determined to be compliant with the applicable standards, regulations, and directives for the countries where the product is marketed. The product is a xed with regulatory marking and text as necessary for the country/agency. Generally, Information Technology Equipment (ITE) product compliance is based on IEC and CISPR standards and their national equivalent such as Product Safety, IEC 60950-1 and European Norm EN 60950-1 or EMC, CISPR 22/CISPR 24 and EN 55022/55024. Dell products have been verified to comply with the EU RoHS Directive 2011/65/EU. Dell products do not contain any of the restricted substances in concentrations and applications not permitted by the RoHS Directive.

8. 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, some functions require conditional tests. All of these tests are listed and described in this section.

8.1. Power-Up Tests

All the power-up self-tests are performed automatically 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
Triple-DES	KATs for ECB and CBC modes: encryption and decryption are tested separately
DSA	KAT: signature generation and verification are tested separately
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-1, SHA-224, SHA-256, SHA-384 and SHA-512	KAT
HMAC-SHA-1, HMAC-SHA-244, HMAC-SHA-256, HMAC-SHA-384 and HMAC-SHA-512	KAT
NIST SP800-90A Hash_DRBG	KAT
Module integrity	DSA signature verification with 2048 bits key and SHA-256

Table 9: Module Self-Tests

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

8.2. Conditional Tests

The following table provides the lists of Pairwise Consistency Test (PCT) 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 using sign/verify for DSA key generation
ECDSA	PCT using sign/verify for ECDSA key generation
RSA	PCT using sign/verify for RSA key generation

Table 10: Module Conditional Tests

9. Guidance

9.1. Crypto Officer Guidance

The version of the RPMs containing the FIPS validated Module is stated in section 1.3. The RPM packages forming the Module can be installed by standard tools recommended for the installation of RPM packages on a Red Hat Enterprise Linux system (for example, yum, rpm, and the RHN remote management tool). All RPM packages are signed with the Red Hat build key, which is an RSA 2048 bit key using SHA-256 signatures. The signature is automatically verified upon installation of the RPM package. If the signature cannot be validated, the RPM tool rejects the installation of the package. In such a case, the Crypto Officer is requested to obtain a new copy of the module's RPMs from Red Hat.

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 1.2 are allowed to be used.

9.1.1. FIPS module installation instructions

Recommended method

The system-wide cryptographic policies package (crypto-policies) contains a tool that completes the installation of cryptographic modules and enables self-checks in accordance with the requirements of Federal Information Processing Standard (FIPS) Publication 140-2. We call this step “FIPS enablement”. The tool named fips-mode-setup installs and enables or disables all the validated FIPS modules and it is the recommended method to install and configure a RHEL-8 system.

1. To switch the system to FIPS enablement in RHEL 8:

```
# fips-mode-setup --enable
Setting system policy to FIPS
FIPS mode will be enabled.
Please reboot the system for the setting to take effect.
```

2. Restart your system:

```
# reboot
```

3. After the restart, you can check the current state:

```
# fips-mode-setup --check
FIPS mode   is enabled.
```

Note: As a side effect of the enablement procedure the fips-mode-enable tool also changes the system-wide cryptographic policy level to a level named “FIPS”, this level helps applications by changing configuration defaults to approved algorithms.

Manual method

The recommended method automatically performs all the necessary steps.
The following steps can be done manually but are not recommended and are not required if the systems has been installed with the fips-mode-setup tool:

- create a file named /etc/system-fips, the contents of this file are never checked

- ensure that the kernel boot line is configured with the fips=1 parameter set
- Reboot the system

NOTE: If `/boot` or `/boot/efi` resides on a separate partition, the kernel parameter `boot=<boot partition>` must be supplied. The partition can be identified with the command "`df | grep boot`". For example:

```
$ df |grep boot
```

/dev/sda1	233191	30454	190296	14%	/boot

The partition of the `/boot` file system is located on `/dev/sda1` in this example.

`Therefore the parameter boot=/dev/sda1 needs to be appended to the kernel command line in addition to the parameter` fips=1

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 into FIPS Approved mode. Failure to do so will not allow the application to properly enter FIPS Approved mode.

9.1.2. 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 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/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).

9.2. 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.

The following module initialization steps must be followed by the Crypto-Officer 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 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 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 3.1.

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 calling application in encrypted form with `FC_UnwrapKey`. The key transport methods allowed for this purpose in FIPS Approved mode is 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.

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.

9.2.1. 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.

9.2.2. RSA and DSA Keys

The Module allows the use of 1024 bits RSA and DSA keys for legacy purposes including signature generation, which is disallowed to be used in FIPS Approved mode as per NIST SP800-131A. Therefore, the cryptographic operations with the non-approved key sizes will result the module operating in non-Approved mode implicitly.

9.2.3. 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.

9.2.4. Key derivation using SP800-132 PBKDF

The module provides password-based key derivation (PBKDF), 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 to [SP800-132] and IG D.6, 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 entered password is acceptable for the users. The minimum value shall be 1000.
- Passwords or passphrases, used as an input for the PBKDF, shall not be used as cryptographic keys.
- The length of the password or passphrase 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].

9.3. Handling Self-Test Errors

When the Module enters the Error state, it needs to be reinitialized to resume normal operation. Reinitialization is accomplished by reinitializing the system.

10. 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

11. Glossary and Abbreviations

AES	Advanced Encryption Specification
AES-NI	Intel Advanced Encryption Standard New Instructions
CAVP	Cryptographic Algorithm Validation Program
CBC	Cypher Block Chaining
CSP	Critical Security Parameter
CTR	Counter Block Chaining
CVL	Component Validation List
DES	Data Encryption Standard
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
ECDSA	Elliptic Curve Digital Signature Algorithm
GCM	Galois/Counter Mode
HMAC	Hash Message Authentication Code
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
O/S	Operating System
PKCS	Public-Key Cryptography Standards
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
TLS	Transport layer Security

12. References

[1] FIPS 140-2 Standard, https://csrc.nist.gov/projects/cryptographic-module-validation-program/standards

[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://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[5] FIPS 180-4 Secure Hash Standard, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[6] FIPS 198-1 The Keyed-Hash Message Authentication Code (HMAC), https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf

[7] FIPS 186-4 Digital Signature Standard (DSS), http://csrc.nist.gov/publications/PubsFIPS.hhttps://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[8] NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation: Methods and Techniques, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[9] NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf

[10] NIST SP 800-38F, Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf

[11] NIST SP 800-56A Revision 3, Recommendation for Pair-Wise Key Establishment Schemes using Discrete Logarithm Cryptography (Revised), https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf

[12] NIST SP 800-67 Revision 2, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf

[13] NIST SP 800-90A Revision 1, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

[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.