



# Red Hat

**Red Hat Enterprise Linux 8 NSS Cryptographic Module  
version rhel8.20201215  
FIPS 140-2 Non-Proprietary Security Policy**

Document Version 1.3  
Last Update: 2022-10-19

## Table of Contents

1. Cryptographic Module Specification.....	4
1.1. Description of the Module.....	4
1.2. Description of the Approved Modes.....	5
1.3. Cryptographic Boundary.....	12
1.3.1. Hardware Block Diagram.....	13
1.3.1. Hardware Block Diagram.....	13
1.3.2. Software Block Diagram.....	14
1.3.2. Software Block Diagram.....	14
2. Cryptographic Module Ports and Interfaces.....	15
2.1. PKCS #11.....	15
2.2. Inhibition of Data Output.....	15
2.3. Disconnecting the Output Data Path from the Key Processes.....	16
3. Roles, Services and Authentication.....	17
3.1. Roles.....	17
3.2. Role Assumption.....	17
3.3. Strength of Authentication Mechanism.....	17
3.4. Multiple Concurrent Operators.....	18
3.5. Services.....	18
3.5.1. Calling Convention of API Functions.....	18
3.5.1. Calling Convention of API Functions.....	18
3.5.2. API Functions.....	18
3.5.2. API Functions.....	18
4. Physical Security.....	27
5. Operational Environment.....	28
5.1 Applicability.....	28
5.2 Policy.....	28
6. Cryptographic Key Management.....	29
6.1. Random Number Generation.....	32
6.2. Key/CSP Storage.....	33
6.3. Key Establishment.....	33
6.4. Key/CSP Zeroization.....	34
6.5. Key Derivation.....	34
7. Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC).....	35
7.1 Statement of compliance.....	35
8. Self-Tests.....	36
8.1. Power-Up Tests.....	36
8.2. Conditional Tests.....	38
9. Guidance.....	39
9.1. Crypto Officer Guidance.....	39
9.1.1. FIPS module installation instructions.....	39
9.1.1. FIPS module installation instructions.....	39
9.1.2. Access to Audit Data.....	40
9.1.2. Access to Audit Data.....	40
9.2. User Guidance.....	40
9.2.1. RSA and DSA Keys.....	41
9.2.1. RSA and DSA Keys.....	41
9.2.2. Triple-DES Keys.....	41
9.2.2. Triple-DES Keys.....	41
9.2.3. Key derivation using SP800-132 PBKDF.....	41
9.2.3. Key derivation using SP800-132 PBKDF.....	41
9.2.4. AES-GCM IV.....	42

9.2.4. AES-GCM IV..... 42

9.3. Handling Self-Test Errors..... 42

10. Mitigation of Other Attacks..... 43

11. Glossary and Abbreviations..... 44

12. References..... 45

# 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 rhel8.20201215 (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 set of libraries designed to support cross-platform development of security-enabled client and server applications. Applications built with NSS can support SSL v2 and v3, TLS, IKE, PKCS #5, PKCS #7, PKCS #11, 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 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 R440	Intel(R) Xeon(R) Silver 4216	Red Hat Enterprise Linux 8	Yes	Yes

Table 2: Tested Platforms

**NOTE:** This validation is only for the tested platform listed in Table 2 of this document. It does not cover other derivatives of the Operating Systems (I.e, CentOS or Fedora).

The Module has been tested for the following configurations:

- 64-bit library, x86\_64 with and without AES-NI enabled.

To operate the Module, the operating system must be restricted to a single operator mode of operation. (This should not be confused with single user mode which is run level 1 on Red Hat Enterprise Linux (RHEL). This refers to processes having access to the same cryptographic instance which RHEL ensures this cannot happen by the memory management hardware.)

The following platform has not been tested as part of the FIPS 140-2 level 1 certification however Red Hat “vendor affirms” that this platform is equivalent to the tested and validated platform. Additionally, Red Hat affirms that the module will function the same way and provide the same security services on any of the systems listed below.

Manufacturer	Processor	O/S & Ver.	Tested
Dell PowerEdge R430	Intel(R) Xeon(R) E5	Red Hat Enterprise Linux 8	N/A

Table 2A: Vendor Affirmed Operating Environment

*Per FIPS 140-2 IG G.5, the 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.*

## 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	CAVP Certificate
Encryption and decryption	AES encryption and decryption with ECB, CBC, CTR and GCM modes	AES 128, 192 and 256 bits keys	Certs. #A1173, #A1174
	AES encryption and decryption with CMAC mode		Cert. #A1176
	AES key wrapping and unwrapping with KW and KWP modes		Certs. #A1175 #A1177
	Three-key Triple-DES encryption and decryption with ECB and CBC modes	Three-key Triple-DES 192 bits keys	Cert. #A1173
Message digest	SHA-1 <sup>1</sup> , SHA-224, SHA-256, SHA-384 and SHA-512	N/A	Cert. #A1173
	HMAC with SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512	At least 112 bits HMAC keys	
Random number generation	NIST SP800-90A Hash_DRBG with SHA-256	Entropy input string, seed, V and C values	Cert. #A1173
Signature generation and verification	DSA signature generation	L=3072, N=256 (with SHA-224, SHA-256, SHA-384, SHA-512)	Cert. #A1173
		L=2048, N=256 (with SHA-224, SHA-256 SHA-384, SHA-512)	
		L=2048, N=224 (with SHA-224, SHA-256 SHA-384, SHA-512)	
	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 SHA-1, SHA-224, SHA-256, SHA-384, SHA-512)	
		L=1024, N=160 (with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512)	

1 SHA-1 is used in the approved mode for secure hash algorithm, HMAC, DSA Signature Verification, ECDSA Signature Verification and PBKDF key derivation only.

Usage	Approved Algorithm	Keys/CSPs	CAVP 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)	
	FIPS 186-4 RSA signature generation according to PKCS#1v1.5	RSA 2048, 3072 and 4096 bits keys (with SHA-256, SHA-384, SHA-512)	
	FIPS 186-4 RSA signature verification according to PKCS#1v1.5	RSA 2048, 3072 and 4096 bits keys (with SHA-1, SHA-256, SHA-384, SHA-512)	
Key and domain parameter generation	FIPS 186-4 RSA key pair generation	RSA 2048, 3072 and 4096 bits keys	Cert. #A1173
	DSA key pair generation	L=3072, N=256	
		L=2048, N=256	
		L=2048, N=224	
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	
	DSA domain parameter verification	L=3072, N=256 (with SHA-256) L=2048, N=256 (with SHA-256) L=2048, N=224 (with SHA-224) L=1024, N=160 (with SHA-1)	
Encryption padding	RSA OAEP Partial_Val	Modulo 2048, 3072, 4096, 6144, 8192 with SHA-224, SHA-256, SHA-384, SHA-512	Cert. #A1173

Usage	Approved Algorithm	Keys/CSPs	CAVP Certificate
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)
	SP 800-56Arev3 Safe Primes Key Generation  Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192  for Diffie-Hellman shared secret computation	Diffie-Hellman key pair	Cert. #A1173
Key derivation	SP800-135 key derivation TLS v1.0, TLS v1.1, TLS v1.2 SHA2-256, SHA2-384, SHA2-512	TLS pre-master secret and master secret	CVL Cert. #A1173
	SP800-56Crev1 KDA HKDF for TLS1.3 PRF.  HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512		Cert. #A1172
	SP800-135 key derivation IKE v1 and v2	IKE SA encryption and integrity keys, IPsec SA encryption and integrity keys, shared secret	CVL Cert. #A1178
	SP800-132 PBKDF	PBKDF password PBKDF derived key	Cert. #A1173



Usage	Approved Algorithm	Keys/CSPs	CAVP Certificate
Shared Secret Computation <sup>2</sup>	SP 800-56Arev3 KAS-FFC-SSC  dhEphem Scheme with safe prime groups for Diffie-Hellman Shared Secret Computation  Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Diffie-Hellman key pair  Shared secret	Cert. #A1173
	SP 800-56Arev3 KAS-ECC-SSC  ECC Ephemeral Unified Scheme for EC Diffie-Hellman Shared Secret Computation  NIST curves P-256, P-384, P-521	EC Diffie-Hellman key pair  Shared secret	
KTS	SP800-38F AES KW, KWP	AES keys 128, 192, 256 bits	Certs. #A1175 #A1177
	SP800-38F AES GCM		Certs. #A1173, #A1174
	SP800-38F AES CBC and HMAC	AES keys 128, 256 bits	AES Certs. #A1173, #A1174  HMAC Cert. #A1173
	SP800-38F Triple-DES CBC and HMAC		Triple-Des Cert. #A1173  HMAC Cert. #A1173
	RFC 3560 RSA OAEP	RSA keys 2048, 3072, 4096, 6144, 8192 bits	Cert. #A1173
Entropy	SP800-90B ENT (NP)	N/A	N/A

<sup>2</sup> KAS-FFC-SSC and KAS-ECC-SSC components which are not SP 800-56Arev3, although tested by CAVP, are not used by the module and only the SP 800-56Arev3 compliant are used.

*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 PKCS#1-v1.5 key wrapping (encrypt, decrypt)	RSA key size between 2048 bits and 16384 bits (or more)	Allowed according to IG D.9

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

Table 5 lists the non-Approved algorithms, use of these algorithms will result the module operating in non-Approved mode implicitly.

Usage	non-Approved Algorithm
Encryption and decryption	AES CTS
	AES-GCM with counter IV generation
	AES-GCM with random IV generation using less than 96 bits
	AES-GCM with external IV generation
	Camellia
	Chacha20/Poly1305 AEAD
	DES
	RC2
	RC4
	RC5
	SEED
Signature generation and verification	Two-key Triple-DES encryption/decryption
	DSA signature generation with key size not equal to 2048 or 3072 bits or with SHA-1. DSA signature verification with key size not equal to 1024, 2048 or 3072 bits
	RSA signature generation with key size not equal to 2048, 3072 or 4096 bits or with SHA-1. RSA signature generation and signature verification using SHA-224 RSA signature verification with key size not equal to 1024, 2048, 3072 or 4096 bits
	ECDSA signature generation using SHA-1

Usage	non-Approved Algorithm
Key Derivation	PBKDF (non-compliant with SP800-132)
Key Derivation	KBKDF (no KAT)
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
	RSA key pair generation for key sizes not listed in Table 3
	AES/Triple-DES non-SP 800-38F compliant key wrapping
	Diffie-Hellman shared secret computation with key size less than 2048 bits or use of non safe primes
	Diffie-Hellman keys generated with domain parameters other than safe primes.
	RSA key wrapping (encrypt, decrypt) with key size less than 2048 bits
	J-PAKE key agreement
EC with Edwards 25519 Curve	

*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 3.53.1-17.el8\_3, which contains the following files:
  - /usr/lib64/libsoftokn3.chk (64 bits)
  - /usr/lib64/libsoftokn3.so (64 bits)
- nss-softokn-freebl RPM with version 3.53.1-17.el8\_3, which contains the following files:
  - /usr/lib64/libfreeblpriv3.chk (64 bits)
  - /usr/lib64/libfreeblpriv3.so (64 bits)

### 1.3.1. Hardware Block Diagram

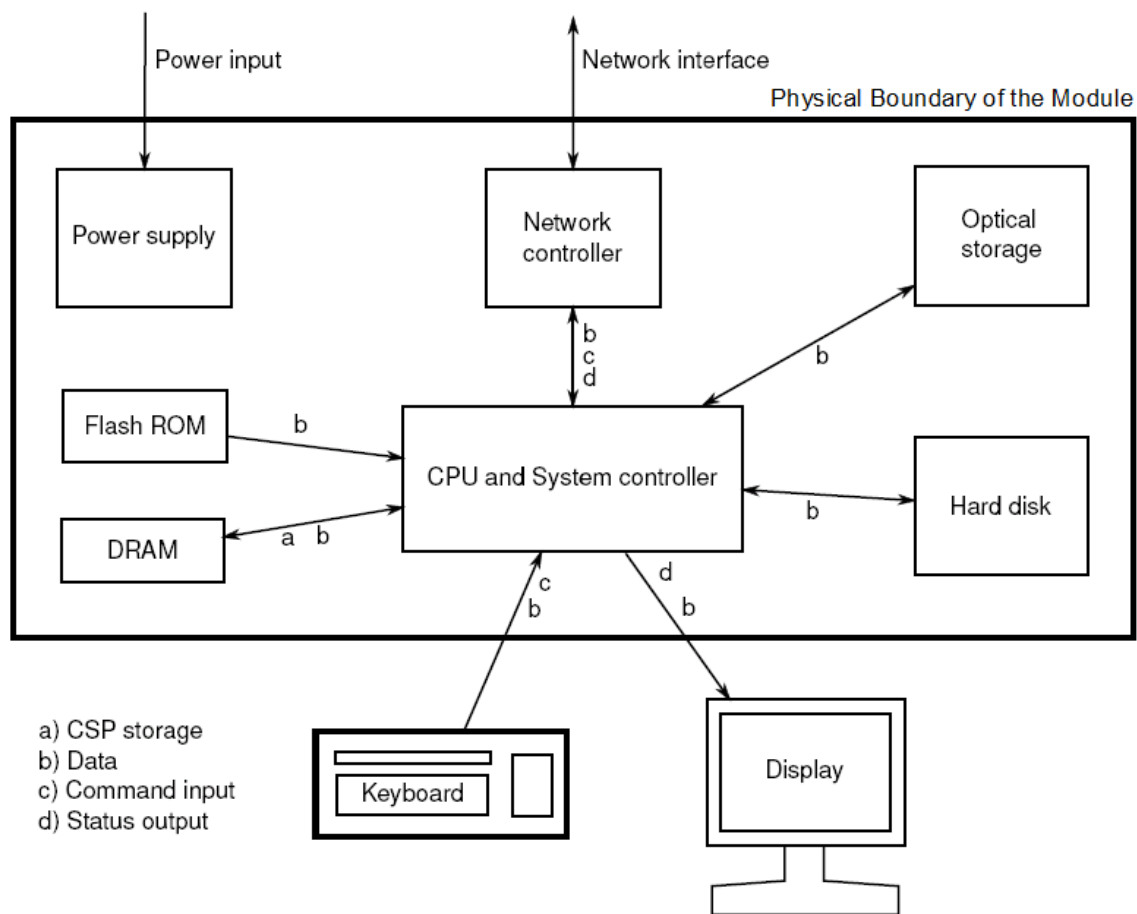


Figure 1: Hardware Block Diagram

### 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.

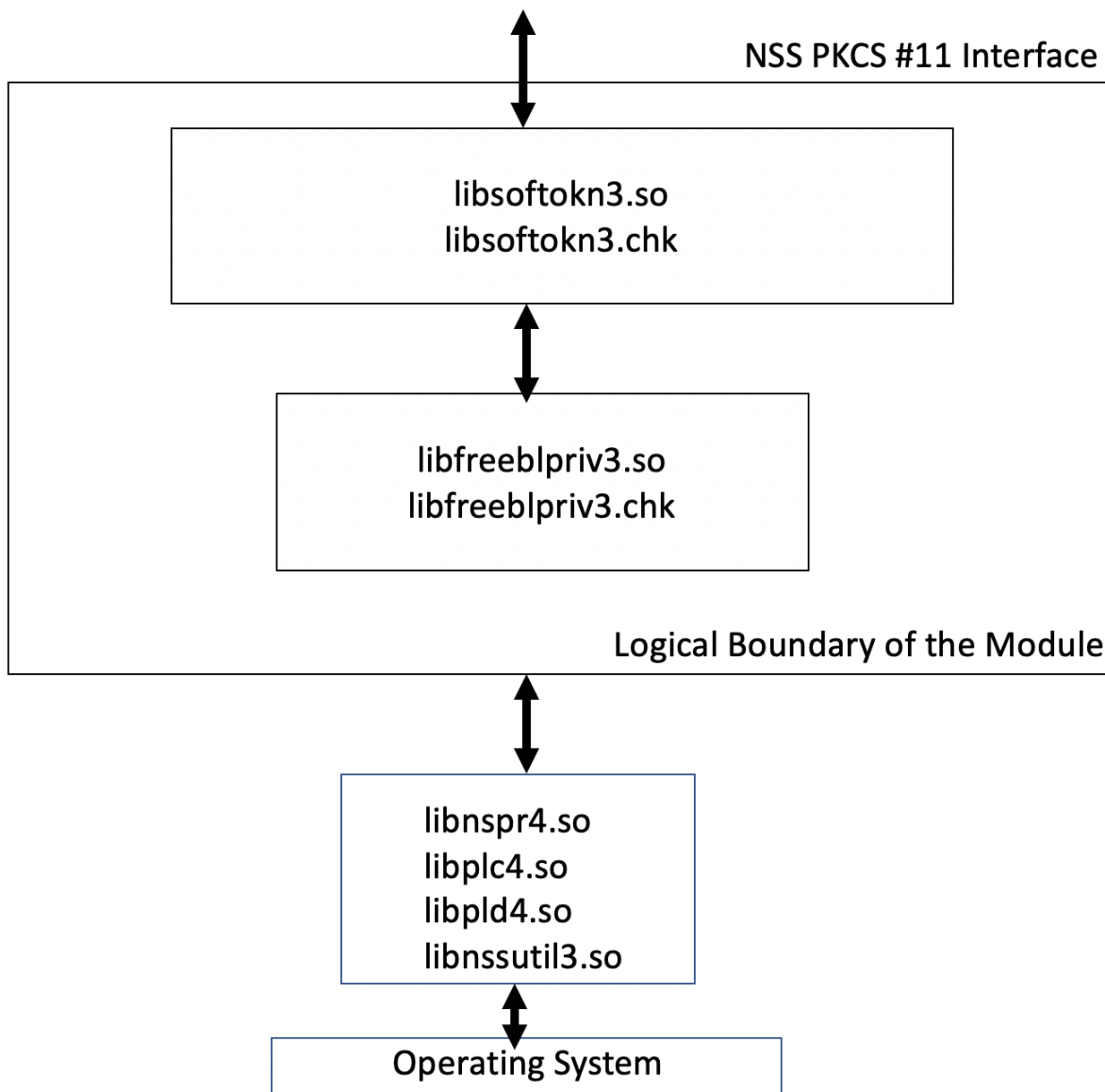


Figure 2: Software Block Diagram

## 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 zeroes 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 3 and Table 4 for the Approved or allowed cryptographic algorithms supported by the Module.

**Note:** The message digest 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	N/A
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	N/A
	CO	FC_Finalize	Finalize (shut down) the module library	All keys	Z
	CO	FC_GetInfo	Obtain general information about the module library	none	N/A
Slot and token management	CO	FC_GetSlotList	Obtain a list of slots in the system	none	N/A
	CO	FC_GetSlotInfo	Obtain information about a particular slot	none	N/A
	CO	FC_GetTokenInfo	Obtain information about the token (This function provides the Show Status service)	none	N/A
	CO	FC_GetMechanismList	Obtain a list of mechanisms (cryptographic algorithms) supported by a token	none	N/A

Service	Role	Function	Description	Keys/CSPs	Access
	CO	FC_GetMechanismInfo	Obtain information about a particular mechanism	none	N/A
	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	N/A
	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	N/A
	CO	FC_GetOperationState	Save the state of the cryptographic operations in a session (This function is only implemented for message digest operations)	none	N/A
	CO	FC_SetOperationState	Restore the state of the cryptographic operations in a session (This function is only implemented for message digest operations)	none	N/A
	U	FC_Login	Log into a token	User password	R
	U	FC_Logout	Log out from a token	none	N/A
Object management	U	FC_CreateObject	Create a new object	Any key type	W
	U	FC_CopyObject	Create a copy of an object	Any key type	R W
	U	FC_DestroyObject	Destroy an object	Any key type	Z
	U	FC_GetObjectSize	Obtain the size of an object in bytes	Any key type	R
	U	FC_GetAttributeValue	Obtain an attribute value of an object	Any key type	R

Service	Role	Function	Description	Keys/CSPs	Access
	U	FC_SetAttributeValue	Modify an attribute value of an object	Any key type	W
	U	FC_FindObjectsInit	Initialize an object search operation	none	N/A
	U	FC_FindObjects	Continue an object search operation	Any key type matching the search criteria	R
	U	FC_FindObjectsFinal	Finish an object search operation	none	N/A
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	N/A
	CO	FC_Digest	Digest single-part data	none	N/A
	CO	FC_DigestUpdate	Continue a multiple-part digesting operation	none	N/A
	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	N/A
Signature generation and	U	FC_SignInit	Initialize a signature operation	DSA/ECDSA/RSA private key, HMAC key	R

Service	Role	Function	Description	Keys/CSPs	Access
verification	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, HMAC key	R
	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

Service	Role	Function	Description	Keys/CSPs	Access
	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_DecryptVerifyUpdate	Continue a multiple-part decryption and verify operation	DSA/ECDSA/RSA public key, HMAC key	R
				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	W
	U	FC_GenerateKeyPair	Generate a public/private key pair (This function performs the pair-wise consistency tests)	RSA/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: AES(KW, KWP) or RSA encryption	AES key, RSA public key, wrapped key(of any key type)	R
					R
	U	FC_UnwrapKey	Unwrap (decrypt) a key using the following mechanism: AES(KW, KWP) or RSA decryption	AES key, RSA private key, wrapped key(of any key type)	R
					W
U	FC_DeriveKey	Shared secret computation	Diffie-Hellman/EC Diffie-Hellman public and private components	R	
				Diffie-Hellman/EC Diffie-Hellman shared secrets	W

Service	Role	Function	Description	Keys/CSPs	Access
			Derive a key from TLS master secret which is derived from TLS pre-master secret	TLS pre-master secret	R
				TLS master secret	RW
			Derive a key from DH/ECDH shared secret using IKE KDF	TLS and HKDF Derived keys	RW
				IKE shared secret (from DH/ECDH SSC)	W
			Derive a key from DH/ECDH shared secret using HKDF	IKE SA Tunnel Encryption Keys	W
				IKE SA Tunnel Integrity Keys	
				IPsec SA Tunnel Encryption Keys	
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 not parallel)	none	N/A
	CO	FC_CancelFunction	A legacy function, which simply returns the value 0x00000051 (function not parallel)	none	N/A
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	All secret or	Z



Service	Role	Function	Description	Keys/CSPs	Access
	CO	FC_InitToken FC_Finalize FC_CloseSession FC_CloseAllSessions	automatically zeroized when freeing the cipher handle	private keys and password	

Table 7: Services details in FIPS Approved mode

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 listed in Table 5, 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 listed in Table 5, Camellia, ChaCha20/Poly1305 AEAD, 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	
Signature generation and verification	FC_SignInit	DSA signature generation with non-compliant key size listed in Table 5, DSA signature generation with SHA-1, RSA signature generation with key size not equal to 2048, 3072 or 4096 bits or with SHA-1, RSA signature generation and signature verification using SHA-224, RSA signature verification with key size not equal to 1024, 2048, 3072 or 4096 bits, ECDSA signature generation with SHA-1
	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	

Service	Function	non-Approved Algorithm invoked
	FC_VerifyRecoverInit	
	FC_VerifyRecover	
Dual-function cryptographic operations	FC_DigestEncryptUpdate	MD2, MD5, AES CTS mode, AES-GCM listed in Table 5, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES, Chacha20/Poly1305
	FC_DecryptDigestUpdate	AES CTS mode, AES-GCM listed in Table 5, Camellia, DES, RC2, RC4, RC5, SEED, MD2, MD5, Two-key Triple-DES, Chacha20/Poly1305
	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, AES-GCM listed in Table 5, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES, ChaCha20/Poly1305
	FC_DecryptVerifyUpdate	AES CTS mode, AES-GCM listed in Table 5, 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, Two-key Triple-DES, ChaCha20/Poly1305
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/Triple-DES non-SP 800-38F compliant key wrapping (encrypt), Triple-DES key wrapping (encrypt) using Two-key Triple-DES, RSA key wrapping (encrypt) with non-compliant key size listed in Table 5
	FC_UnwrapKey	AES/Triple-DES non-SP 800-38F compliant key wrapping (decrypt), 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 shared secret computation with non-compliant key size listed in Table 5, Diffie-Hellman keys generated with domain parameters other than safe primes, EC with Edwards 25519 Curve, J-PAKE key agreement, PBKDF (non-compliant with SP800-132), KBKDF

*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 1.1.

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.

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 192 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 public and private key with key size 2048 and 3072 bits	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 public and private key for curve 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 public and private key with key size 2048, 3072 and 4096	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 FC_UnwrapKey for input and FC_WrapKey for output	Automatically zeroized when freeing the cipher handle

DRBG entropy input string	Obtained from CPU jitter source which is the ENT(NP)	Application memory	N/A	Automatically zeroized when freeing DRBG handle
DRBG seed, 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	N/A	Application memory	Entry: API input parameter Output: N/A	Automatically zeroized when freeing the cipher handle
TLS master secret	Derived from TLS pre-master secret by using key derivation	Application memory	N/A	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	Application memory	Encrypted for output using FC_WrapKey	Automatically zeroized when freeing the cipher handle
HKDF derived keys	Derived SP800-56Crev1 HKDF KDF mechanisms	Application memory	Encrypted for output using FC_WrapKey	Automatically zeroized when freeing the cipher handle
Diffie-Hellman public and private key	Public and private keys are generated using the SP800-56ARev3 Safe Primes key generation method. Random values are obtained from the SP800-90A DRBG.	Application memory	Encrypted through key wrapping using FC_UnwrapKey for input and FC_WrapKey for output	Automatically zeroized when freeing the cipher handle
EC Diffie-Hellman public and private key	Public and private keys are generated using the FIPS 186-4 key generation method. Random values are obtained from the SP800-90A DRBG.	Application memory	Encrypted through key wrapping using FC_UnwrapKey for input and FC_WrapKey for output	Automatically zeroized when freeing the cipher handle

Diffie-Hellman shared secret	Generated during shared secret computation.	Application memory	Encrypted for output using FC_WrapKey	Automatically zeroized when freeing the cipher handle
EC Diffie-Hellman shared secret	Generated during shared secret computation.	Application memory	Encrypted for output using FC_WrapKey	Automatically zeroized when freeing the cipher handle
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	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 Derived Keys	Derived SP800-135 IKE KDF mechanisms	Application memory	Encrypted for output using FC_WrapKey	Automatically zeroized when freeing the cipher handle
IPsec SA Tunnel Encryption 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
IPsec SA Tunnel Integrity 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

PKDF 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

### 6.1. Random Number Generation

The Module provides a Hash based SP800-90A-compliant Deterministic Random Bit Generator (DRBG) with SHA-256 for creation of key components of asymmetric keys, and random number generation. The seeding (and automatic reseeding) of the DRBG is done with `getrandom()` call. 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.

Module's DRBG is seeded with an entropy source from the kernel that consists of CPU Jitter noise source and a HMAC\_DRBG conditioning component. The entropy source ENT (NP) is compliant with [SP 800-90B] and is within the module's physical boundary but outside of the module's logical boundary. The CPU Jitter noise source provides 330 bits of entropy to the HMAC\_DRBG, which preserves it at the output. However, this HMAC\_DRBG conditioning component does not implement prediction resistance. Therefore the caveat, "The module generates cryptographic keys whose strengths are modified by available entropy" applies.

The Key Generation methods implemented in the module for Approved services in FIPS mode is compliant with [SP800-133]. The module generates symmetric key through the `FC_GenerateKey()` function using the random numbers from the SP 800-90A DRBG.

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 keys used in the EC Diffie-Hellman shared secret computation schemes are generated internally by the module using the ECDSA key generation method compliant with [FIPS186-4] and [SP800-56Arev3]. The Diffie-Hellman shared secret computation scheme is also compliant with [SP800-56Arev3], and generates keys using safe primes defined in RFC7919 and RFC3526, as described in section 6.3.



## 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 file key4.db) is within the Module's physical boundary but outside its logical boundary.

## 6.3. Key Establishment

The module provides Diffie-Hellman shared secret computation (KAS FFC SSC) and EC Diffie-Hellman shared secret computation (KAS ECC SSC) compliant with SP800-56Arev3, in accordance with scenario X1 (1) of IG D.8. The module provides support for Diffie-Hellman and EC Diffie-Hellman key agreement schemes compliant with SP800-56Arev3 by offering separate services for shared secret computation and the key derivation using the SP800-135 TLS KDF (for use within the TLS protocol), so that the user application can implement the key agreement.

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. Note that the module only implements key generation and shared secret computation of safe primes, and no other part of the TLS protocol (with the exception of the TLS KDF, which is separately implemented).

- 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)

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.

- RSA key wrapping with PKCS#1-v1.5 provides between 112 and 256 bits of encryption strength; Allowed per IG D.9
- AES key wrapping with KW, KWP and GCM key establishment methodology provides between provides between 128 and 256 bits of encryption strength.
- AES key wrapping with AES CBC and HMAC key establishment methodology provides 128 or 256 bits of encryption strength).
- Triple-DES Key wrapping with Triple-DES CBC and HMAC key establishment methodology provides 112 bits of encryption strength).
- RSA key wrapping with OAEP key establishment methodology provides between 112 and 200 bits of encryption strength).

## 6.4. Key/CSP Zeroization

The application that uses the Module is responsible for appropriate zeroization of the key material. The Module provides zeroization methods to clear the memory region previously occupied by a plaintext secret key, private key or password. A plaintext secret or private key 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.

## 6.5. 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 (PBKDF). The implementation is compliant with option 1a of [SP-800-132]. Keys derived from passwords or passphrases using this method can only be used in storage applications.

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

MARKETING NAME..... PowerEdge R440  
REGULATORY MODEL..... E45S  
REGULATORY TYPE..... E45S001  
EFFECTIVE DATE..... March 01, 2020  
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 affixed 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, CBC, CMAC and GCM modes: encryption and decryption are tested separately
Triple-DES	KATs for ECB and CBC modes: encryption and decryption are tested separately
DSA	KAT: DSA signature generation and verification with L=2048, N=224 and SHA-224 (separately tested).
ECDSA	KAT: ECDSA signature generation and verification with P-256 and SHA-256 (separately tested).
Diffie-Hellman (KAS-FFC-SSC)	Primitive "Z" Computation KAT with 2048-bit key
EC Diffie-Hellman (KAS_ECC_SSC)	Primitive "Z" Computation KAT with P-256 curve
RSA	KAT: RSA with 2048-bit key, public key encryption and private key decryption (separately tested).  KAT: RSA PKCS#1 v1.5 signature generation and verification with 2048-bit key and SHA-256, SHA-384 and SHA-512 (separately tested).
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
TLS KDF	KAT: TLS 1.0 PRF KAT and TLS 1.2 KAT using SHA-224, SHA-256, SHA-384 and SHA-512

Algorithm	Test
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.
PBKDF KDF	KAT: Using SHA-256
NIST SP800-90A Hash_DRBG	KAT
SP 800-90-A DRBG	Health test per section 11.3 of SP 800-90A DRBG
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 Error: Reference source not found. 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-setup 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 configure the operational environment to run the FIPS validated module. Failure to do so will not allow the application to properly use the FIPS validated module.

### 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 a private key database format: key4.db. "key4.db" format is based on SQL DataBase engine and can be used concurrently by multiple processes. The database is considered outside the Module's logical boundary and all data stored in the database 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. 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.

### 9.2.2. 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.

### 9.2.3. 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], 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].

#### 9.2.4. 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

The module also complies with IG A.5, provision 2. The GCM random IV is generated by using the approved `Hash_DRBG` and the user must ensure that the IV length is at least 96 bits.

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

### 9.3. Handling Self-Test Errors

When the Module enters the Error state, it needs to be reinitialized to resume normal operation. Reinitialization of the module is accomplished by rebooting 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	<p><b>RSA blinding</b></p> <p>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.</p>	None
Cache-timing attacks on the modular exponentiation operation used in RSA and DSA	<p><b>Cache invariant modular exponentiation</b></p> <p>This is a variant of a modular exponentiation implementation that Colin Percival [17] showed to defend against cache-timing attacks</p>	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	<p><b>Double-checking RSA signatures</b></p> <p>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.</p>	None

*Table 11: Mitigation of other attacks*

## 11. Glossary and Abbreviations

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