Red Hat Enterprise Linux 9 NSS Cryptographic Module

version 4.34.0-a20cd33fbbe14357

FIPS 140-3 Non-Proprietary Security Policy

document version 1.2

Last update: 2024-08-20

Prepared by:

atsec information security corporation

4516 Seton Center Pkwy, Suite 250

Austin, TX 78759

www.atsec.com

Table of Contents

1 General

1.1 Overview

This document is the non-proprietary FIPS 140-3 Security Policy for version 4.34.0-a20cd33fbbe14357 of the Red Hat Enterprise Linux 9 NSS Cryptographic Module. It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS PUB 140-3 for an overall Security Level 1 module. This Non-Proprietary Security Policy may be reproduced and distributed, but only whole and intact and including this notice. Other documentation is proprietary to their authors.

1.2 How this Security Policy was Prepared

In preparing the Security Policy document, the laboratory formatted the vendor-supplied documentation for consolidation without altering the technical statements therein contained. The further refining of the Security Policy document was conducted iteratively throughout the conformance testing, wherein the Security Policy was submitted to the vendor, who would then edit, modify, and add technical contents. The vendor would also supply additional documentation, which the laboratory formatted into the existing Security Policy, and resubmitted to the vendor for their final editing.

1.3 Security Levels

Table 1 describes the individual security areas and levels of FIPS 140-3.

ISO/IEC 24759 Section 6. [Number Below]	FIPS 140-3 Section Title	Security Level
1	General	1
2	Cryptographic Module Specification	1
3	Cryptographic Module Interfaces	1
4	Roles, Services, and Authentication	1
5	Software/Firmware Security	1
6	Operational Environment	1
7	Physical Security	Not Applicable
8	Non-invasive Security	Not Applicable
9	Sensitive Security Parameter Management	1
10	Self-tests	1
11	Life-cycle Assurance	1
12	Mitigation of Other Attacks	1

Table 1 - Security Levels

2 Cryptographic Module Specification

2.1 Description

The Red Hat Enterprise Linux 9 NSS Cryptographic Module (hereafter referred to as “the module”) is defined as a software module in a multi-chip standalone embodiment. It provides a C language application program interface (API) designed to support cross-platform development of security-enabled client and server applications. Applications built with NSS can support SSLv3, TLS, IKEv2, PKCS#5, PKCS#7, PKCS#11, PKCS#12, S/MIME, X.509 v3 certificates, and other security standards supporting FIPS 140-3 validated cryptographic algorithms. It combines a vertical stack of Linux components intended to limit the external interface each separate component may provide.

2.2 Operational Environments

The module has been tested on the following platforms with the corresponding module variants and configuration options with and without PAA/PAI:

#	Operating System	Hardware Platform	Processor	PAA/ Acceleration
1	Red Hat Enterprise Linux 9	Dell PowerEdge R440	Intel(R) Xeon(R) Silver 4216	AES-NI
2	Red Hat Enterprise Linux 9	IBM z16 3931-A01	IBM z16	CPACF
3	Red Hat Enterprise Linux 9	IBM 9080 HEX	IBM POWER10	ISA

Table 2 - Tested Operational Environments

In addition to the configurations tested by the atsec CST laboratory, vendor affirmed testing was performed on the following platforms for the module by Red Hat. The Red Hat Enterprise Linux operating system is used as the basis of other products which include but are not limited to

#	Operating System	Hardware Platform
1	Red Hat Enterprise Linux 9	Intel(R) Xeon(R) E5

Table 3 - Vendor Affirmed Products

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

2.3 Approved Algorithms

Table 4 lists all approved cryptographic algorithms of the module, including specific key lengths employed for approved services (Table 9), and implemented modes or methods of operation of the algorithms.

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A3463	SHA [FIPS 180-4]	SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message digest
A3463 A3470	AES [FIPS 197, SP 800-38A]	ECB, CBC, CTR	128, 192, 256 bits	Encryption Decryption
A3468	AES [FIPS 197, SP 800-38A, SP 800-38A Addendum]	CBC-CS1	128, 192, 256 bits	Encryption Decryption
A3463 A3470 A4482	AES [FIPS 197, SP 800-38D]	GCM (internal IV)	128, 192, 256 bits	Encryption
A3463 A3470 A4482	AES [FIPS 197, SP 800-38D]	GCM (external IV)	128, 192, 256 bits	Decryption
A3464 A3469	AES [FIPS 197, SP 800-38F]	KW, KWP	128, 192, 256 bits	Key wrapping Key unwrapping
A3465	AES [FIPS 197, SP 800-38B]	CMAC	128, 192, 256 bits	Message authentication
A3463	HMAC [FIPS 198-1]	SHA-224, SHA-256, SHA-384, SHA-512	112-256 bits	Message authentication
A3466	KBKDF [SP 800-108r1]	Counter, feedback, and double pipeline mode, using CMAC and HMAC SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112-256 bits	Key derivation
A3462	HKDF [SP 800-56Cr2]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112-256 bits	Key derivation
A3463	TLS 1.0/1.1 KDF [SP 800-135r1] (CVL)	MD5-SHA1	112-256 bits	Key derivation
A3463	TLS 1.2 KDF [SP 800-135r1] (CVL)	SHA-256, SHA-384, SHA-512	112-256 bits	Key derivation
A3467	IKEv2 PRF [SP 800-135r1] (CVL)	SHA-1, SHA-256, SHA-384, SHA-512	112-256 bits	Key derivation
A3463	PBKDF2 [SP 800-132]	Option 1a with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112-256 bits	Password-based key derivation
A3463	Hash_DRBG [SP 800-90Ar1]	SHA-256	256 bits	Random number generation
A3463	KAS-FFC-SSC [SP 800-56Ar3]	dhEphem (initiator/responder)	MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192, ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192 (112-200 bits)	Shared secret computation
A3463	KAS-ECC-SSC [SP 800-56Ar3]	Ephemeral Unified Model (initiator/responder)	P-256, P-384, P-521 (128, 192, 256 bits)	Shared secret computation

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A3463	RSA [FIPS 186-4]	PKCS#1 v1.5 and PSS with SHA-224, SHA-256, SHA-384, SHA-512	2048-4096 bits (112-150 bits)	Signature generation
A3463	RSA [FIPS 186-4]		2048-4096 bits (112-150 bits)	Signature verification
A3463	RSA [FIPS 186-2]		1024-1536 bits (80-97 bits)	Signature verification (legacy algorithm)
A3463	DSA [FIPS 186-4]	SHA-224, SHA-256, SHA-384, SHA-512	L = 1024, N= 160 L = 2048, N = 224 L = 2048, N = 256 L = 3072, N = 256 (80-128 bits)	Signature verification (legacy algorithm)
A3463	ECDSA [FIPS 186-4]	SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521 (128, 192, 256 bits)	Signature generation
A3463	ECDSA [FIPS 186-4]			Signature verification
A3463	Safe primes [SP 800-56Ar3]	SP 800-56Ar3 Section 5.6.1.1.4 Testing Candidates	MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192, ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192 (112-200 bits)	Key pair generation
A3463	RSA [FIPS 186-4]	FIPS 186-4 Appendix B.3.3 Probable Primes	2048-4096 bits (112-150 bits)	Key pair generation
A3463	ECDSA [FIPS 186-4]	FIPS 186-4 Appendix B.4.1 Extra Random Bits	P-256, P-384, P-521 (128, 192, 256 bits)	Key pair generation
Vendor affirmed	CKG [SP 800-133r2]	Direct generation (using the SP 800-90Ar1 DRBG)	112-256 bits	Secret key generation
		Safe primes	MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192, ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192 (112-200 bits)	Key pair generation
		RSA	2048-16384 bits (112-256 bits)	
		ECDSA	P-256, P-384, P-521 (128, 192, 256 bits)	

Table 4 - Approved Algorithms

2.4 Non-Approved Algorithms Allowed in the Approved Mode of Operation with no Security Claimed

Table 5 lists the non-approved algorithms that are allowed in the approved mode of operation with no security claimed. These algorithms are used by the approved services listed in Table 9.

Algorithm / Functions	Caveat	Use / Function

MD5	Only allowed as the PRF in TLSv1.0 and v1.1 per IG 2.4.A. No security claimed.	Message digest used in TLS v1.0/1.1 KDF only

Table 5 - Non-Approved Algorithms Allowed in the Approved Mode of Operation with No Security Claimed

2.5 Non-Approved Algorithms Not Allowed in the Approved Mode of Operation

The module does not offer any non-approved cryptographic algorithms that are allowed in approved services with security claimed.

Table 6 lists all non-approved cryptographic algorithms of the module employed by the non-approved services in Table 10.

Algorithm / Functions	Use / Function
MD2, MD5, SHA-1	Message digest
RC2, RC4, DES, Triple-DES, CDMF, Camellia, SEED, ChaCha20(-Poly1305)	Encryption Decryption
AES GCM (external IV)	Encryption
CBC-MAC, AES XCBC-MAC, AES XCBC-MAC-96	Message authentication
HMAC (MD2, MD5, SHA-1; < 112-bit keys)	
HMAC/SSLv3 MAC (constant-time implementation)	
MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, DES, Triple-DES, AES, Camellia, SEED, ANS X9.63 KDF, SSL 3 PRF, IKEv1 PRF	Key derivation
KBKDF, HKDF, TLS 1.0/1.1 KDF, TLS 1.2 KDF, IKEv2 PRF (< 112-bit keys)	
KBKDF (MD2, MD5)	
IKEv2 PRF (MD2, MD5)	
PKCS#5 PBE, PKCS#12 PBE	Password-based key derivation
PBKDF2 (password length < 8 characters, salt length < 128 bits, iteration count < 1000, or key length < 112 bits)	
J-PAKE	Shared secret computation
Diffie-Hellman shared secret computation (FIPS 186-type groups)	
EC Diffie-Hellman shared secret computation (P-192)	
DSA	Signature generation
RSA (primitive; PKCS#1 v1.5 or PSS with MD2, MD5)	Signature generation Signature verification
ECDSA (P-192)	
RSA	Key encapsulation Key un-encapsulation
DSA	Parameter generation Parameter verification Key pair generation

DH (FIPS 186-type groups)	Key pair generation
RSA (< 2048 bits)	
ECDSA (P-192)	
Symmetric key generation (< 112 bits)	Secret key generation

Table 6 - Non-Approved Algorithms Not Allowed in the Approved Mode of Operation

2.6 Module Design and Components

Figure 1 shows a block diagram that represents the design of the module when the module is operational and providing services to other user space applications. In this diagram, the physical perimeter of the operational environment (a general-purpose computer on which the module is installed) is indicated by a purple dashed line. The cryptographic boundary is represented by the components painted in orange blocks, which consists of two software components:

1. The Softoken library, which provides a PKCS#11 token API (libsoftokn3.so), and its associated integrity check value (libsoftokn3.chk).
2. The Freebl cryptographic library, which implements most cryptographic algorithms used by Softoken (libfreeblpriv3.so), and its associated integrity check value (libfreeblpriv3.chk).

Green lines indicate the flow of data between the cryptographic module and its operator application, through the logical interfaces defined in Section 3.

Components in white are only included in the diagram for informational purposes. They are not included in the cryptographic boundary (and therefore not part of the module’s validation). For example, the kernel is responsible for managing system calls issued by the module itself, as well as other applications using the module for cryptographic services.

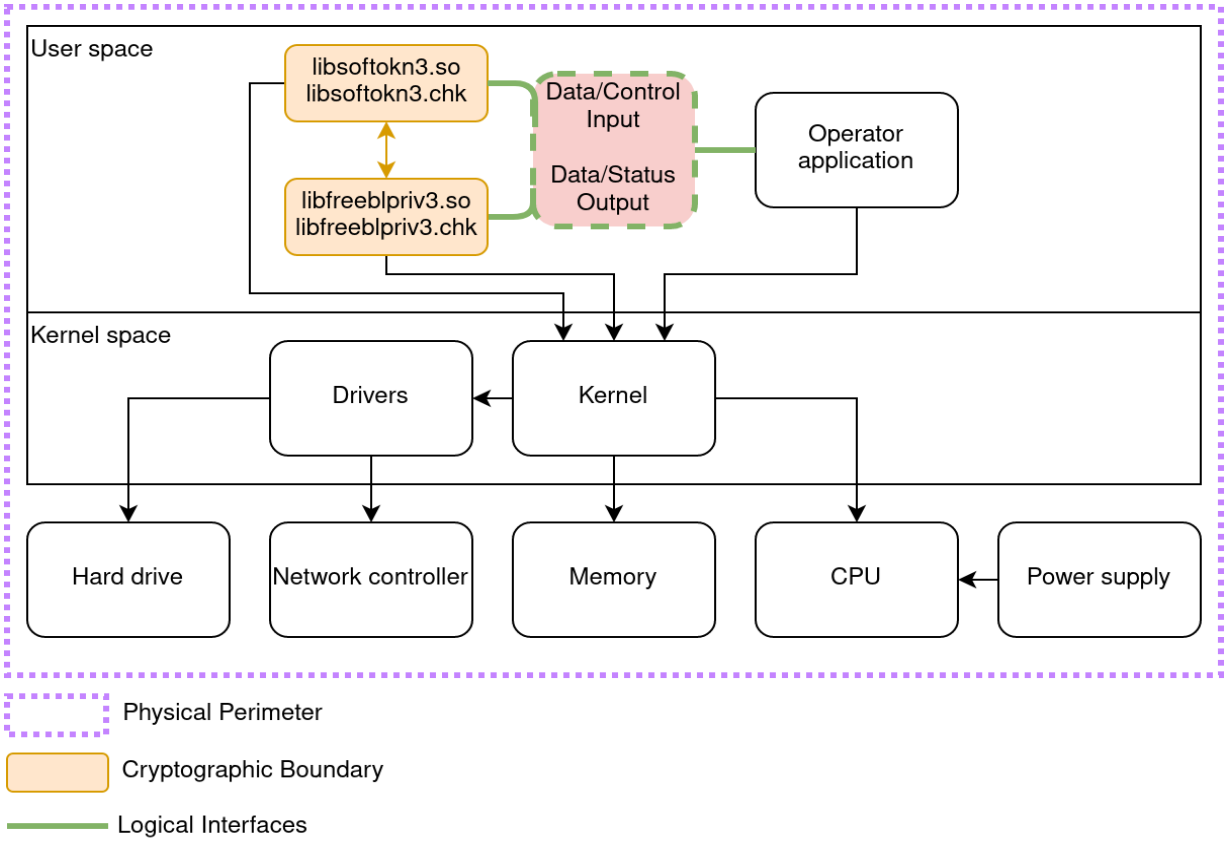

Figure 1 – Software Block Diagram

2.7 Rules of Operation

Upon initialization, the module immediately performs all Freebl cryptographic algorithm self-tests (CASTs) as specified in Table 13. When all those self-tests pass successfully, the module automatically performs the pre-operational integrity test on the libfreeblpriv3.so file using its associated check value.

Then, the module performs the RSA CAST in the Softoken library, followed by the the pre-operational integrity test on the libsoftokn3.so file using its associated check value. Finally, all remaining CASTs for the algorithms implemented in Softoken are executed (see Table 13).

Only if all CASTs and pre-operational integrity tests passed successfully, the module transitions to the operational state. No operator intervention is required to reach this point.

In the operational state, the module accepts service requests from calling applications through its logical interfaces. At any point in the operational state, a calling application can end its process, thus causing the module to end its operation.

The module supports two modes of operation:

- The approved mode of operation, in which the approved or vendor affirmed services are available as specified in Table 9. By default, the module is operating in the approved mode of operation after passing all CASTs and pre-operational integrity tests. The module can be transitioned from the approved mode of operation to the non-approved mode of operation by requesting one of the non-approved services specified in Table 10.
- The non-approved mode of operation, in which the non-approved services are available as specified in Table 10. The module can be transitioned from the non-approved mode of

operation to the approved mode of operation by requesting one of the approved services specified in Table 9.

2.8 Industry Protocols

The module supports the IKEv2 KDF, TLS 1.0/1.1 KDF, and TLS 1.2 KDF. No parts of the IKEv2 or TLS protocols, other than the approved cryptographic algorithms and the KDFs, have been tested by the CAVP and CMVP.

3 Cryptographic Module Ports and Interfaces

The logical interfaces are the APIs through which the applications request services. These logical interfaces are logically separated from each other by the API design. Note that this API corresponds to the functionality described in the PKCS#11 standard (Cryptographic Token Interface Current Mechanisms Specification). All data output via data output interface is inhibited when the module is performing pre-operational test, conditional cryptographic algorithm self-tests, zeroization, or when the module enters the error state. Table 7 summarizes the logical interfaces:

Physical Port	Logical Interface	Data that passes over port / interface
As a software-only module, the module does not have physical ports. Physical Ports are interpreted to be the physical ports of the hardware platform on which it runs.	Data Input	API input parameters
	Data Output	API output parameters
	Control Input	API function calls, API input parameters for control input
	Status Output	API return codes, error queue

Table 7 - Ports and Interfaces

The module does not implement a control output interface.

4 Roles, services, and authentication

4.1 Roles

The module supports the Crypto Officer role only. This sole role is implicitly and always assumed by the operator of the module. No support is provided for multiple concurrent operators or a maintenance role.

Table 8 lists the roles supported by the module with corresponding services with input and output parameters.

Role	Service	Input	Output
Crypto Officer	Message digest	Message	Digest value
	Encryption	Plaintext, AES key	Ciphertext
	Decryption	Ciphertext, AES key	Plaintext
	Key wrapping	AES key, any CSP except for password	Wrapped CSP
	Key unwrapping	AES key, wrapped CSP	Any CSP except for password
	Message authentication	Message, AES key or HMAC key	MAC tag
	Message authentication verification	Message, AES key or HMAC key, MAC tag	Pass/fail
	Key derivation	Key-derivation key or shared secret	KBKDF derived key, HKDF derived key, TLS derived key, or IKEv2 derived key
	Password-based key derivation	Password	PBKDF2 derived key
	Random number generation	Output length	Random bytes
	Shared secret computation	Owner private key, peer public key	Shared secret
	Signature generation	Message, private key, hash algorithm	Signature
	Signature verification	Message, public key, signature, hash algorithm	Pass/fail
	Key encapsulation	Plaintext, public key	Ciphertext
	Key un-encapsulation	Ciphertext, private key	Plaintext
	Parameter generation	Parameter size	Domain parameters
	Parameter verification	Domain parameters	Pass/fail
	Key pair generation	Key size	Key pair
	Secret key generation	Key size	AES key, HMAC key, or Key-derivation key
	Show version	N/A	Name and version information
	Show status	N/A	Module status
	Self-test	N/A	Pass/fail results of self-tests
	Zeroization	Any SSP	N/A

Table 8 - Roles, Service Commands, Input and Output

4.2 Authentication

The module does not support authentication for roles.

4.3 Services

The module provides services to operators that assume the available role. All services are described in detail in the API documentation (manual pages). The next tables define the services that utilize approved and non-approved security functions in this module. For the respective tables, the convention below applies when specifying the access permissions (types) that the service has for each SSP.

- **Generate (G)**: The module generates or derives the SSP.
- **Read (R)**: The SSP is read from the module (e.g. the SSP is output).
- **Write (W)**: The SSP is updated, imported, or written to the module.
- **Execute (E)**: The module uses the SSP in performing a cryptographic operation.
- **Zeroize (Z)**: The module zeroizes the SSP.
- **N/A**: The module does not access any SSP or key during its operation.

To interact with the module, a calling application must use the FIPS token APIs provided by Softoken. The FIPS token API layer can be used to retrieve the approved service indicator for the module. This indicator consists of four independent service indicators:

1. The session indicator, which must be used for all cryptographic services except the key derivation service. It can be accessed by invoking the NSC_NSSGetFIPSStatus function with the CKT_NSS_SESSION_LAST_CHECK parameter. If the output parameter is set to CKS_NSS_FIPS_OK (1), the service was approved.
2. The object indicator, which must be used for the key derivation service. It can be accessed by invoking the NSC_NSSGetFIPSStatus function with the CKT_NSS_OBJECT_CHECK parameter and the output derived key. If the output parameter is set to CKS_NSS_FIPS_OK (1), the service was approved.
3. The DRBG service indicator, which must be used for the DRBG service. It can be accessed by invoking the C_SeedRandom or C_GenerateRandom functions. If any of these functions returns CKR_OK, the service was approved.
4. The DSA signature verification indicator, which must be used for the DSA signature verification service. It can be accessed by invoking the C_VerifyInit function with any CKM_DSA_* mechanism parameter. If this function returns CKR_OK, the service was approved.

Table 9 lists the approved services in this module, the algorithms involved, the Sensitive Security Parameters (SSPs) involved and how they are accessed, the roles that can request the service, and the respective service indicator. In this table, CO specifies the Crypto Officer role.

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Message digest	Compute a message digest	SHA-224, SHA-256, SHA-384, SHA-512	N/A	CO	N/A	CKS_NSS_FIPS_OK
Encryption	Encrypt a plaintext	AES-ECB, AES-CBC, AES-CBC-CTS-CS1, AES-CTR, AES-GCM (internal IV)	AES key	CO	W, E	

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Decryption	Decrypt a ciphertext	AES-ECB, AES-CBC, AES-CBC-CTS-CS1, AES-CTR, AES-GCM (external IV)		CO	W, E	
Key wrapping	Wrap a CSP	AES-KW, AES-KWP	AES key Any CSP except for password	CO	W, E	
Key unwrapping	Unwrap a CSP		AES key	CO	W, E	
			Any CSP except for password		G, R	
Message authentication	Compute a MAC tag	AES-CMAC HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	AES key HMAC key	CO	W, E	
Message authentication verification	Verify a MAC tag	AES-CMAC HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	AES key HMAC key	CO	W, E	
Key derivation	Derive a key from a key-derivation key or a shared secret	KBKDF	Key-derivation key	CO	W, E	
			KBKDF derived key		G, R	
		HKDF, TLS 1.0/1.1 KDF, TLS 1.2 KDF, IKEv2 KDF	Shared secret		W, E	
			HKDF derived key, TLS derived key, IKEv2 derived key		G, R	
Password-based key derivation	Derive a key from a password	PBKDF2	Password	CO	W, E	
			PBKDF2 derived key		G, R	
Random number generation	Generate random bytes	Hash_DRBG	Entropy input	CO	W, E	CKR_OK
			DRBG seed		E, G	
			Internal state (V, C)		W, E, G	
Shared secret computation	Compute a shared secret	KAS-FFC-SSC	DH private key (owner), DH public key (peer)	CO	W, E	CKS_NSS_FIPS_OK
			Shared secret		G, R	
		KAS-ECC-SSC	EC private key (owner), EC public key (peer)		W, E	
			Shared secret		G, R	
Signature generation	Generate a signature	RSA signature generation (PKCS#1 v1.5 and PSS)	RSA private key	CO	W, E	
		ECDSA signature generation	EC private key			
Signature verification	Verify a signature	RSA signature verification (PKCS#1 v1.5 and PSS)	RSA public key	CO	W, E	
		ECDSA signature verification	EC public key			

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
		DSA signature verification	DSA public key			CKR_OK
Key pair generation	Generate a key pair	CKG Hash_DRBG Safe primes key pair generation RSA key pair generation ECDSA key pair generation	DH private key, DH public key RSA private key, RSA public key EC private key, EC public key	CO	G, R	CKS_NSS_FIPS_OK
			Intermediate key generation value		G, E, Z	
			Internal state (V, C)		W, E	
Secret key generation	Generate a secret key	CKG Hash_DRBG	AES key HMAC key Key-derivation key	CO	G, R	
			Internal state (V, C)		W, E	
Show version	Return the name and version information	N/A	N/A	CO	N/A	None
Show status	Return the module status	N/A	N/A	CO	N/A	None
Self-test	Perform the CASTs and integrity test	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 AES-GCM, AES-ECB, AES-CBC, AES-CMAC HMAC KBKDF HKDF TLS 1.0/1.1 KDF TLS 1.2 KDF IKEv2 PRF PBKDF2 Hash_DRBG KAS-FFC-SSC KAS-ECC-SSC RSA DSA ECDSA See Table 13 for specifics	N/A	CO	N/A	None
Zeroization	Zeroize all SSPs	N/A	Any SSP	CO	Z	None

Table 9 - Approved Services

Table 10 lists the non-approved services in this module, the algorithms involved, the roles that can request the service, and the respective service indicator. In this table, CO specifies the Crypto Officer role.

Service	Description	Algorithms Accessed	Role	Indicator
Message digest	Compute a message digest	MD2, MD5, SHA-1	CO	N/A
Encryption	Encrypt a plaintext	RC2, RC4, DES, Triple-DES, CDMF, Camellia, SEED, ChaCha20(-Poly1305) AES GCM (external IV)	CO	N/A
Decryption	Decrypt a ciphertext	RC2, RC4, DES, Triple-DES, CDMF, Camellia, SEED, ChaCha20(-Poly1305)	CO	N/A
Message authentication	Compute a MAC tag	CBC-MAC, AES XCBC-MAC, AES XCBC-MAC-96 HMAC (MD2, MD5, SHA-1; < 112-bit keys) HMAC/SSLv3 MAC (constant-time implementation)	CO	N/A
Key derivation	Derive a key from a key-derivation key or a shared secret	MD2, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, DES, Triple-DES, AES, Camellia, SEED, ANS X9.63 KDF, SSL 3 PRF, IKEv1 PRF KBKDF, HKDF, TLS 1.0/1.1 KDF, TLS 1.2 KDF, IKEv2 PRF (< 112-bit keys) KBKDF (MD2, MD5) IKEv2 PRF (MD2, MD5)	CO	N/A
Password-based key derivation	Derive a key from a password	PKCS#5 PBE, PKCS#12 PBE PBKDF2 (password length < 8 characters, salt length < 128 bits, iteration count < 1000, or key length < 112 bits)	CO	N/A
Shared secret computation	Compute a shared secret	J-PAKE Diffie-Hellman shared secret computation (FIPS 186-type groups) EC Diffie-Hellman shared secret computation (P-192)	CO	N/A
Signature generation	Generate a signature	DSA signature generation RSA signature generation (primitive; PKCS#1 v1.5 or PSS with MD2, MD5) ECDSA signature generation (P-192)	CO	N/A
Signature verification	Verify a signature	RSA signature verification (primitive; PKCS#1 v1.5 or PSS with MD2, MD5) ECDSA signature verification (P-192)	CO	N/A
Key encapsulation	Encapsulate a key	RSA encapsulation	CO	N/A
Key un-encapsulation	Un-encapsulate a key	RSA un-encapsulation	CO	N/A
Parameter generation	Generate domain parameters	DSA parameter generation	CO	N/A
Parameter verification	Verify domain parameters	DSA parameter verification	CO	N/A
Key pair generation	Generate a key pair	DH key pair generation (FIPS 186-type groups) RSA key pair generation (< 2048 bits) DSA key pair generation ECDSA key pair generation (P-192)	CO	N/A
Secret key generation	Generate a secret key	Symmetric key generation (< 112 bits)	CO	N/A

Table 10 - Non-Approved Services

5 Software/Firmware security

5.1 Integrity Techniques

The integrity of the module is verified by performing DSA signature verification with a 2048-bit key and SHA-256. Each software component of the module has an associated integrity check value, which contains the DSA signature of the shared library.

5.2 On-Demand Integrity Test

Integrity tests are performed as part of the pre-operational self-tests, which are executed when the module is initialized. The integrity tests may be invoked on-demand by unloading and subsequently re-initializing the module, which will perform (among others) the software integrity tests.

6 Operational Environment

6.1 Applicability

The module operates in a modifiable operational environment per FIPS 140-3 level 1 specification: the module executes on a general purpose operating system (Red Hat Enterprise Linux 9), which allows modification, loading, and execution of software that is not part of the validated module.

6.2 Tested Operational Environments

See 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 Ansible Automation Platform
- Red Hat OpenStack Platform
- Red Hat OpenShift
- Red Hat Gluster Storage
- Red Hat Satellite

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

6.3 Policy and Requirements

The module shall be installed as stated in Section 11.2. If properly installed, the operating system provides process isolation and memory protection mechanisms that ensure appropriate separation for memory access among the processes on the system. Each process has control over its own data and uncontrolled access to the data of other processes is prevented.

There are no concurrent operators.

The module does not have the capability of loading software or firmware from an external source.

Instrumentation tools like the `ptrace` system call, `gdb` and `strace`, userspace live patching, as well as other tracing mechanisms offered by the Linux environment such as `ftrace` or `systemtap`, shall not be used in the operational environment. The use of any of these tools implies that the cryptographic module is running in a non-validated operational environment.

7 Physical Security

The module is comprised of software only and therefore this section is not applicable.

8 Non-invasive Security

This module does not implement any non-invasive security mechanism and therefore this section is not applicable.

9 Sensitive Security Parameters Management

Table 10 summarizes the Sensitive Security Parameters (SSPs) that are used by the cryptographic services implemented in the module in the approved services (Table 9).

Each SSP will have a parameter (`isFIPS`), which indicates whether this SSP was established in an approved manner or not. This ensures separation of CSPs between approved and non-approved services.

Key / SSP Name / Type	Strength	Security Function and Cert. Number	Generation	Import / Export	Establishment	Storage	Zeroization	Use and related keys
AES key (CSP)	128, 192, 256 bits	AES (A3463, A3468, A3470, A4482) AES CMAC (A3465)	Hash_DRBG (A3463) (SP 800-133r2 Section 6.1)	Imported in wrapped form Exported in wrapped form	N/A	RAM	C_DestroyObject, Module reset	**Use**: Encryption, Decryption, Key wrapping, Key unwrapping, Message authentication, **Related SSPs**: Internal state (V, C)
HMAC key (CSP)	112-256 bits	HMAC (A3463)	Hash_DRBG (A3463) (SP 800-133r2 Section 6.1)	Imported in wrapped form Exported in wrapped form	N/A	RAM	C_DestroyObject, Module reset	**Use**: Message authentication **Related SSPs**: Internal state (V, C)
Key-derivation key (CSP)	112-256 bits	KBKDF (A3466)	Hash_DRBG (A3463) (SP 800-133r2 Section 6.1)	Imported in wrapped form Exported in wrapped form	N/A	RAM	C_DestroyObject, Module reset	**Use**: Key derivation **Related SSPs**: Internal state (V, C), KBKDF derived key
Shared secret (CSP)	112-256 bits	KAS-FFC-SSC (A3463) KAS-ECC-SSC (A3463) HKDF (A3462) TLS 1.0/1.1 KDF (A3463) TLS 1.2 KDF (A3463) IKEv2 PRF (A3467)	N/A	Imported in wrapped form Exported in wrapped form	KAS-FFC-SSC, KAS-ECC-SSC	RAM	C_DestroyObject, Module reset	**Use**: Shared secret computation, Key derivation **Related SSPs**: DH public key, DH private key, EC public key, EC private key, HKDF derived key, TLS derived key, IKEv2 derived key
Password (CSP)	N/A	PBKDF2 (A3463)	N/A	Imported in plaintext form No export	N/A	RAM	Module reset	**Use**: Password-based key derivation **Related SSPs**: PBKDF2 derived key
KBKDF derived key (CSP)	112-256 bits	KBKDF (A3466)	KBKDF (A3466) (SP 800-133r2 Section 6.2)	No import Exported in wrapped form	N/A	RAM	C_DestroyObject, Module reset	**Use**: Key derivation

								Related SSPs: Key-derivation key
HKDF derived key (CSP)	112-256 bits	HKDF (A3462)	HKDF (A3462) (SP 800-133r2 Section 6.2)	No import Exported in wrapped form	N/A	RAM	C_DestroyObject, Module reset	**Use**: Key derivation **Related SSPs**: Shared secret
TLS derived key (CSP)	112-256 bits	TLS 1.0/1.1 KDF (A3463) TLS 1.2 KDF (A3463)	TLS 1.0/1.1 KDF (A3463) TLS 1.2 KDF (A3463) (SP 800-133r2 Section 6.2)	No import Exported in wrapped form	N/A	RAM	C_DestroyObject, Module reset	**Use**: Key derivation **Related SSPs**: Shared secret
IKEv2 derived key (CSP)	112-256 bits	IKEv2 PRF (A3467)	IKEv2 PRF (A3467) (SP 800-133r2 Section 6.2)	No import Exported in wrapped form	N/A	RAM	C_DestroyObject, Module reset	**Use**: Key derivation **Related SSPs**: Shared secret
PBKDF2 derived key (CSP)	112-256 bits	PBKDF2 (A3463)	PBKDF2 (A3463) (SP 800-133r2 Section 6.2)	No import Exported in wrapped form	N/A	RAM	C_DestroyObject, Module reset	**Use**: Password-based key derivation **Related SSPs**: Password
Entropy input (CSP) (per IG D.L)	256 bits at initial seeding, 225 bits at reseeding	Hash_DRBG (A3463)	RHEL Userspace CPU Time Jitter RNG Entropy Source (ESV cert. #47)	No import No export	N/A	RAM	Automatic, Module reset	**Use**: Random number generation **Related SSPs**: DRBG seed
DRBG seed (CSP) (per IG D.L)	256 bits	Hash_DRBG (A3463)	Hash_DRBG (A3463)	No import No export	N/A	RAM	Automatic, Module reset	**Use**: Random number generation **Related SSPs**: Entropy input, Internal state (V, C)
Internal state (V, C) (CSP) (per IG D.L)	256 bits	Hash_DRBG (A3463)	Hash_DRBG (A3463)	No import No export	N/A	RAM	Module reset	**Use**: Random number generation **Related SSPs**: DRBG seed
DH private key (CSP)	112-200 bits	KAS-FFC-SSC (A3463)	SP 800-56Ar3 (safe primes) Section 5.6.1.1.4 Testing Candidates	Imported in wrapped form Exported in wrapped form	N/A	RAM	C_DestroyObject, Module reset	**Use**: Shared secret computation **Related SSPs**: Shared secret, Internal state (V, C), DH public key
DH public key (PSP)	112-200 bits			Imported in plaintext form Exported in plaintext form				**Use**: Shared secret computation **Related SSPs**: Shared secret, Internal state (V, C), DH private key
EC private key (CSP)	112, 128, 192, 256 bits	KAS-ECC-SSC (A3463) ECDSA (A3463)	FIPS 186-4 Appendix B.4.1 Extra Random Bits	Imported in wrapped form Exported in wrapped form	N/A	RAM	C_DestroyObject, Module reset	**Use**: Signature generation, Shared secret

								computation **Related SSPs**: Shared secret, Internal state (V, C), EC public key
EC public key (PSP)	112, 128, 192, 256 bits			Imported in plaintext form Exported in plaintext form				**Use**: Signature verification, Shared secret computation **Related SSPs**: Shared secret, Internal state (V, C), EC private key
RSA private key (CSP)	112-150 bits	RSA (A3463)	FIPS 186-4 Appendix B.3.3 Probable Primes	Imported in wrapped form Exported in wrapped form	N/A	RAM	C_DestroyObject, Module reset	**Use**: Signature generation **Related SSPs**: Internal state (V, C), RSA public key
RSA public key (PSP)	80-150 bits			Imported in plaintext form Exported in plaintext form				**Use**: Signature verification **Related SSPs**: Internal state (V, C), RSA private key
DSA public key (PSP)	80, 112, 128 bits	DSA (A3463)	N/A	Imported in plaintext form No export	N/A	RAM	C_DestroyObject, Module reset	**Use**: Signature verification **Related SSPs**: None
Intermediate key generation value (CSP)	112-256 bits	CKG (vendor affirmed)	SP 800-133r2	No import No export	N/A	RAM	Automatic, Module reset	**Use**: Key pair generation **Related SSPs**: DH public key, DH private key, EC public key, EC private key, RSA public key, RSA private key

Table 11 - SSPs

9.1 Random Bit Generators

The module employs a Deterministic Random Bit Generator (DRBG) implementation based on SP 800-90Ar1. This DRBG is used internally by the module (e.g. to generate symmetric keys, seeds for asymmetric key pairs, and random numbers for security functions). It can also be accessed using the specified API functions.

The DRBG implemented is a SHA-256 Hash_DRBG, seeded by the entropy source described in Table 12. The DRBG is seeded with 384 bits of output from the entropy source and is reseeded with 256 bits of output from the entropy source. There are 0.87890625 bits of entropy per bit of output of the entropy source. The Hash_DRBG does not employ prediction resistance.

The public use document of this entropy source is found at:

https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/entropy/E47_PublicUse.pdf

Entropy Source	Minimum number of bits of entropy	Details
RHEL Userspace CPU Time Jitter RNG Entropy Source (ESV cert. #E47)	225 bits of entropy in the 256-bit output	Userspace CPU Jitter 2.2.0 entropy source is located within the physical perimeter of the module but outside the cryptographic boundary of the module. This entropy source is non-physical.

Table 12 - Non-Deterministic Random Number Generation Specification

As the highest SSP strength required by the module is 256 bits, the following caveat is applicable: "The module generates SSPs (e.g., keys) whose strengths are modified by available entropy."

9.2 SSP Generation

The module implements Cryptographic Key Generation (CKG, vendor affirmed), compliant with SP 800-133r2. When random values are required, they are obtained from the SP 800-90Ar1 approved DRBG, compliant with Section 4 of SP 800-133r2. The following methods are implemented:

- Direct generation of symmetric keys: compliant with SP 800-133r2, Section 4, without the use of V (direct DRBG output as described in additional comment #2 of IG D.H).
- Safe primes key pair generation: the method described in Section 5.6.1.1.4 of SP 800-56Ar3 (“Testing Candidates”) is used. The random values used in key generation are obtained in compliance with SP 800-133r2, Section 4, without the use of V (direct DRBG output as described in additional comment #2 of IG D.H).
- RSA key pair generation: the method described in Appendix B.3.3 of FIPS 186-4 (“Probable Primes”) is used. The random values used in key generation are obtained in compliance with SP 800-133r2, Section 4, without the use of V (direct DRBG output as described in additional comment #2 of IG D.H).
- ECC (ECDH and ECDSA) key pair generation: the method described in Appendix B.4.1 of FIPS 186-4 (“Extra Random Bits”) is used. The random values used in key generation are obtained in compliance with SP 800-133r2, Section 4, without the use of V (direct DRBG output as described in additional comment #2 of IG D.H).

Additionally, the module implements the following key derivation methods:

- KBKDF: compliant with SP 800-108r1. This implementation can be used to generate secret keys from a pre-existing key-derivation-key.
- HKDF: compliant with SP 800-56Cr2. This implementation shall only be used to generate secret keys in the context of an SP 800-56Ar3 key agreement scheme.
- TLS 1.0/1.1 KDF, TLS 1.2 KDF, IKEv2 PRF: compliant with SP 800-135r1. These implementations shall only be used to generate secret keys in the context of the TLS 1.0/1.1, TLS 1.2, and IKEv2 protocols, respectively.
- PBKDF2: compliant with option 1a of SP 800-132. This implementation shall only be used to derive keys for use in storage applications.

Intermediate key generation values are not output from the module and are explicitly zeroized after processing the service.

9.3 SSP Establishment

The module provides Diffie-Hellman (DH) and Elliptic Curve Diffie-Hellman (ECDH) shared secret computation compliant with SP800-56Ar3, in accordance with scenario 2 (1) of FIPS 140-3 IG D.F.

For Diffie-Hellman, the module supports the use of the safe primes defined in RFC 3526 (IKE) and RFC 7919 (TLS). Note that the module only implements domain parameter generation, key pair generation and verification, and shared secret computation. No other part of the IKE or TLS protocols is implemented (with the exception of the TLS 1.0/1.1 KDF, TLS 1.2 KDF, and IKEv2 PRF):

- IKE (RFC 3526):
  - MODP-2048 (ID = 14)
  - MODP-3072 (ID = 15)
  - MODP-4096 (ID = 16)
  - MODP-6144 (ID = 17)
  - MODP-8192 (ID = 18)
- TLS (RFC 7919)
  - ffdhe2048 (ID = 256)
  - ffdhe3072 (ID = 257)
  - ffdhe4096 (ID = 258)
  - ffdhe6144 (ID = 259)
  - ffdhe8192 (ID = 260)

ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, and ffdhe8192 have key sizes of 2048-8192 bits and security strengths of 112-200 bits.

MODP-2048, MODP-3072, MODP-4096, MODP-6144, and MODP-8192 have key sizes of 2048-8192 bits and a security strength of 112-200 bits.

P-256, P-384, and P-521 have have a security strength of 128-256 bits.

To comply with the assurances found in Section 5.6.2 of SP 800-56Ar3, the operator must use the module together with an application that implements the TLS protocol. Additionally, the module's approved "Key pair generation" service must be used to generate ephemeral Diffie-Hellman or EC Diffie-Hellman key pairs, or the key pairs must be obtained from another FIPS-validated module. As part of this service, the module will internally perform the full public key validation of the generated public key. The module's shared secret computation service will internally perform the full public key validation of the peer public key, complying with Sections 5.6.2.2.1 and 5.6.2.2.2 of SP 800-56Ar3.

The module also provides the following key transport mechanisms:

- Key wrapping using AES KW and AES KWP, with a security strength of 128, 192, or 256 bits, depending on the wrapping key size. Compliant with IG D.G.
- Key wrapping using AES GCM with a security strength of 128 or 256 bits. Compliant with IG D.G.

9.4 SSP Entry/Output

The module only supports SSP entry and output to and from the calling application running on the same operational environment. This corresponds to manual distribution, electronic entry/output ("CM Software to/from App via TOEPP Path") per FIPS 140-3 IG 9.5.A Table 1.

CSPs (with the exception of passwords) can only be imported to and exported from the module when they are wrapped using an approved security function (e.g. AES KW or KWP). PSPs can be imported and exported in plaintext. Import and export is performed using API input and output parameters.

9.5 SSP Storage

SSPs imported, generated, derived, or otherwise established by the module are stored in RAM while the module is operational. The operator application can use these SSPs to perform cryptographic operations, or export them as described in Section 9.

The module does not perform persistent storage of SSPs.

9.6 SSP Zeroization

The memory occupied by SSPs is allocated by regular memory allocation operating system calls. The operator application is responsible for calling the appropriate destruction functions provided in the module's API. The destruction functions (listed in Table 11) overwrite the memory occupied by SSPs with zeroes and de-allocate the memory with the regular memory de-allocation operating system call.

10 Self-tests

The module performs pre-operational self-tests and conditional self-tests. While the module is executing the self-tests, services are not available, and data output (via the data output interface) is inhibited until the tests are successfully completed. The module does not return control to the calling application until the tests are completed.

All the self-tests are listed in Table 12, with the respective condition under which those tests are performed. Note that the pre-operational integrity test is only executed after all cryptographic algorithm self-tests (CASTs) executed successfully.

Algorithm	Parameters	Condition	Type	Test
DSA	SHA-256 and 2048-bit key	Initialization (after Freebl CASTs)	Pre-operational Integrity Test	Signature verification on libfreeblpriv3.so file
DSA	SHA-256 and 2048-bit key	Initialization (after RSA CAST)	Pre-operational Integrity Test	Signature verification on libfsoftokn3.so file
SHA-1	N/A	Freebl initialization	Cryptographic Algorithm Self-Test	KAT digest generation
SHA-224	N/A	Freebl initialization	Cryptographic Algorithm Self-Test	KAT digest generation
SHA-256	N/A	Freebl initialization	Cryptographic Algorithm Self-Test	KAT digest generation
SHA-384	N/A	Freebl initialization	Cryptographic Algorithm Self-Test	KAT digest generation
SHA-512	N/A	Freebl initialization	Cryptographic Algorithm Self-Test	KAT digest generation
AES GCM	128, 192, 256-bit key	Freebl initialization	Cryptographic Algorithm Self-Test	KAT encryption and decryption
AES CMAC	128, 192, 256-bit key	Freebl initialization	Cryptographic Algorithm Self-Test	KAT MAC tag generation
AES ECB	128, 192, 256-bit key	Freebl initialization	Cryptographic Algorithm Self-Test	KAT encryption and decryption
AES CBC	128, 192, 256-bit key	Freebl initialization	Cryptographic Algorithm Self-Test	KAT encryption and decryption
HMAC	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	Freebl initialization	Cryptographic Algorithm Self-Test	KAT MAC tag generation
KBKDF	HMAC SHA-256 in counter mode	Softoken initialization	Cryptographic Algorithm Self-Test	KAT key derivation
HKDF	SHA-256	Softoken initialization	Cryptographic Algorithm Self-Test	KAT key derivation
TLS 1.0/1.1 KDF	MD5-SHA1	Freebl initialization	Cryptographic Algorithm Self-Test	KAT key derivation

Algorithm	Parameters	Condition	Type	Test
TLS 1.2 KDF	SHA-256	Freebl initialization	Cryptographic Algorithm Self-Test	KAT key derivation
IKEv2 PRF	SHA-1, SHA-256, SHA-384, SHA-512	Softoken initialization	Cryptographic Algorithm Self-Test	KAT key derivation
PBKDF2	SHA-256 with 5 iterations and 128-bit salt	Softoken initialization	Cryptographic Algorithm Self-Test	KAT password-based key derivation
Hash_DRBG	SHA-256 without prediction resistance	Freebl initialization	Cryptographic Algorithm Self-Test	KAT DRBG generation and reseed
KAS-FFC-SSC	2048-bit key	Freebl initialization	Cryptographic Algorithm Self-Test	KAT shared secret computation
KAS-ECC-SSC	P-256	Freebl initialization	Cryptographic Algorithm Self-Test	KAT shared secret computation
RSA	PKCS#1 v1.5 with SHA-256, SHA-384, SHA-512, and 2048-bit key	Softoken initialization	Cryptographic Algorithm Self-Test	KAT signature generation and verification
DSA	1024-bit key	Freebl initialization	Cryptographic Algorithm Self-Test	KAT signature verification
ECDSA	SHA-256 and P-256	Freebl initialization	Cryptographic Algorithm Self-Test	KAT signature generation and verification
DH	N/A	DH key pair generation	Pair-wise Consistency Test	Section 5.6.2.1.4 pair-wise consistency
ECDH	N/A	EC key pair generation	Pair-wise Consistency Test	Section 5.6.2.1.4 pair-wise consistency
RSA	PKCS#1 v1.5 with SHA-256	RSA key pair generation	Pair-wise Consistency Test	Sign/Verify pair-wise consistency
ECDSA	SHA-256	EC key pair generation	Pair-wise Consistency Test	Sign/Verify pair-wise consistency

Table 13 - Self-Tests

10.1 Pre-operational Tests

The module performs pre-operational tests automatically when the module is powered on. The pre-operational self-tests ensure that the module is not corrupted. The module transitions to the operational state only after the pre-operational self-tests are passed successfully.

The types of pre-operational self-tests are described in the next sub-sections.

10.1.1 Pre-operational Software Integrity Test

The integrity of the module is verified by performing DSA signature verification with a 2048-bit key and SHA-256. Each software component of the module has an associated integrity check value, which contains the DSA signature of the shared library.

If any of the software integrity tests fail, the module transitions to the error state (Section 10.3). As mentioned previously, the DSA and SHA-256 algorithms go through their respective CASTs before the software integrity tests are performed.

The pre-operational integrity test may be invoked on-demand by unloading and subsequently re-initializing the module.

10.2 Conditional Self-Tests

10.2.1 Conditional Cryptographic algorithm tests

The module performs self-tests on all FIPS approved cryptographic algorithms as part of the approved services supported in the approved mode of operation, using the tests shown in Table 13. Data output through the data output interface is inhibited during the self-tests. If any of these tests fails, the module transitions to the error state (Section 10.3).

The conditional self-tests may be invoked on-demand by unloading and subsequently re-initializing the module.

10.2.2 Conditional Pairwise Consistency Test

Upon generation of a DH, RSA or EC key pair, the module will perform a pair-wise consistency test (PCT) as shown in Table 13, which provides some assurance that the generated key pair is well formed. For DH and EC key pairs, these tests consists of the PCT described in Section 5.6.2.1.4 of SP 800-56Ar3. For RSA and EC key pairs, this test consists of a signature generation and a signature verification operation. Note that two PCTs are performed for EC key pairs. If the test fails, the module transitions to the error state (Section 10.3).

10.3 Error States

If the module fails any of the self-tests, the module enters the error state. In the error state, the module immediately stops functioning and ends the application process. Consequently, the data output interface is inhibited, and the module accepts no more inputs or requests (as the module is no longer running).

Table 14 lists the error states and the status indicator values that explain the error that has occurred.

Error State	Cause of Error	Status Indicator
Error	Software integrity test failure	Module will not load
	CAST failure	Module will not load
	PCT failure	Module stops functioning (sftk_fatalError is set to TRUE)

Table 14 - Error States

11 Life-cycle assurance

11.1 Delivery and Operation

The module is distributed through the nss-softokn-3.79.0-18.el9_0 and nss-softokn-freebl-3.79.0-18.el9_0 Red Hat Enterprise Linux 9 RPM packages. The Netscape Portable Runtime (NSPR) package nspr-4.34.0-18.el9_0 is a prerequisite for the module. The NSPR package must be installed in the operating environment. The “Show module name and version” service returns the value `Red Hat Enterprise Linux 9 nss 4.34.0-a20cd33fbbe14357`.

11.1.1 End of life procedures

As the module does not persistently store SSPs, secure sanitization of the module consists of unloading the module. This will zeroize all SSPs in volatile memory. Then, if desired, the nss-softokn-3.79.0-18.el9_0 and nss-softokn-freebl-3.79.0-18.el9_0 RPM packages can be uninstalled from the RHEL 9 system.

11.2 Crypto Officer Guidance

The version of the RPMs containing the FIPS validated Module is stated in section 11.1. 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.

11.2.1 FIPS module installation instructions

Before the nss-softokn-3.79.0-18.el9_0 and nss-softokn-freebl-3.79.0-18.el9_0 RPM packages are installed, the RHEL 9 system must operate in the approved mode. This can be achieved by:

- Adding the `fips=1` option to the kernel command line during the system installation. During the software selection stage, do not install any third-party software. More information can be found at the vendor documentation.
- Switching the system into the approved mode after the installation. Execute the `fips-mode-setup --enable` command. Restart the system. More information can be found at the vendor documentation.

In both cases, the Crypto Officer must verify the RHEL 9 system operates in the approved mode by executing the `fips-mode-setup --check` command.

After installation of the nss-softokn-3.79.0-18.el9_0 and nss-softokn-freebl-3.79.0-18.el9_0 RPM packages, the Crypto Officer must execute the “Show module name and version” service by accessing the CKA_NSS_VALIDATION_MODULE_ID attribute of the CKO_NSS_VALIDATION object in the default slot. The object attribute must contain the value

`Red Hat Enterprise Linux 9 nss 4.34.0-a20cd33fbbe14357`

Alternatively, the /usr/lib64/nss/unsupported-tools/validation tool is provided as a convenience by the nss-tools-3.79.0-18.el9_0 RPM package. This tool performs the same steps, and also outputs the FIPS module identifier as below.

The cryptographic boundary consists only of the Softoken and Freebl libraries along with their associated integrity check values as listed in Section 2.6. If any other NSS API outside of these two libraries is invoked, the user is not interacting with the module specified in this Security Policy.

11.2.2 AES GCM IV

The Crypto Officer shall consider the following requirements and restrictions when using the module.

For TLS 1.2, the module offers the AES GCM implementation and uses the context of Scenario 1 of FIPS 140-3 IG C.H. NSS is compliant with SP 800-52r2 Section 3.3.1 and the mechanism for IV generation is compliant with RFC 5288 and 8446.

The module does not implement the TLS protocol. The module's implementation of AES GCM is used together with an application that runs outside the module's cryptographic boundary. The design of the TLS protocol implicitly ensures that the counter (the nonce_explicit part of the IV) does not exhaust the maximum number of possible values for a given session key.

In the event the module's power is lost and restored, the consuming application must ensure that a new key for use with the AES GCM key encryption or decryption under this scenario shall be established.

Alternatively, the Crypto Officer can use the module's API to perform AES GCM encryption using internal IV generation. These IVs are always 96 bits and generated using the approved DRBG internal to the module's boundary.

Finally, for TLS 1.3, the AES GCM implementation uses the context of Scenario 5 of FIPS 140-3 IG C.H. The protocol that provides this compliance is TLS 1.3, defined in RFC8446 of August 2018, using the cipher-suites that explicitly select AES GCM as the encryption/decryption cipher (Appendix B.4 of RFC8446). The module supports acceptable AES GCM cipher suites from Section 3.3.1 of SP800-52r2. The module's implementation of AES GCM is used together with an application that runs outside the module's cryptographic boundary. The design of the TLS protocol implicitly ensures that the counter (the nonce_explicit part of the IV) does not exhaust the maximum number of possible values for a given session key

11.2.3 Key Derivation using SP 800-132 PBKDF2

The module provides password-based key derivation (PBKDF2), compliant with SP 800-132. The module supports option 1a from Section 5.4 of SP 800-132, in which the Master Key (MK) or a segment of it is used directly as the Data Protection Key (DPK). In accordance to SP 800-132 and FIPS 140-3 IG D.N, the following requirements shall be met:

- Derived keys shall only be used in storage applications. The MK shall not be used for other purposes. The length of the MK or DPK shall be of 112 bits or more.
- Passwords or passphrases, used as an input for the PBKDF2, shall not be used as cryptographic keys.
- The length of the password or passphrase shall be at least 8 characters, and shall consist of lowercase, uppercase, and numeric characters. The probability of guessing the value is estimated to be at most 10^(-8), when all characters are digits. Combined with the minimum iteration count as described below, this provides an acceptable trade-off between user experience and security against brute-force attacks.
- A portion of the salt, with a length of at least 128 bits, shall be generated randomly using the SP 800-90Ar1 DRBG provided by the module.
- 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 is 1000.

11.2.4 AES-XTS

In accordance with IG C.I, the module performs a check to ensure that the two AES-XTS keys, Key_1 and Key_2 are not the same.

To comply with SP800-38E, the length of the data unit for any instance of AES-XTS shall not exceed 2^20 AES blocks.

11.2.5 RSA SigGen and SigVer

The module is compliant with IG C.F. The module supports RSA signature generation with modulus lengths of 2048, 3072, and 4096 bits. All three modulus lengths have been CAVP tested by atsec.

The minimum number of the Miller-Rabin tests used in primality testing are consistent with Appendix B of FIPS 186-4.

The module supports FIPS 186-4 signature verification with modulus lengths of 2048, 3072, and 4096 bits. All three modulus lengths have been CAVP tested by atsec.

The module supports FIPS 186-2 signature verification with modulus lengths of 1024, 1280, 1536 bits. The 1024-bit and 1536-bit modulus lengths have been CAVP tested by atsec. CAVP testing is not available for the 1280-bit modulus length, so it was not CAVP tested.

12 Mitigation of other attacks

The module is designed to mitigate the attacks listed in Table 14.

Attack	Mitigation Mechanism	Specific Limit
Timing attacks on RSA	**RSA blinding** Timing attack on RSA was first demonstrated by Paul Kocher in 1996, who contributed the mitigation code to our module. Most recently Boneh and Brumley 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	**Cache invariant modular exponentiation** This is a variant of a modular exponentiation implementation that Colin Percival showed to defend against cache-timing attacks	This mechanism requires intimate knowledge of the cache line sizes of the processor. The mechanism may be ineffective when the module is running on a processor whose cache line sizes are unknown.
Arithmetic errors in RSA signatures	**Double-checking RSA signatures** Arithmetic errors in RSA signatures might leak the private key. Ferguson and Schneier recommend that every RSA signature generation should verify the signature just generated.	None

Table 15 - Mitigation of other attacks

Appendix A. Glossary and Abbreviations

AES	Advanced Encryption Standard
AES-NI	Advanced Encryption Standard New Instructions
API	Application Programming Interface
CAST	Cryptographic Algorithm Self-Test
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CKG	Cryptographic Key Generation
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CPACF	CP Assist for Cryptographic Functions
CSP	Critical Security Parameter
CTR	Counter
CTS	Ciphertext Stealing
DES	Data Encryption Standard
DH	Diffie-Hellman
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
ENT (NP)	Non-physical Entropy Source
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standards
GCM	Galois Counter Mode
HKDF	HMAC-based Key Derivation Function
HMAC	Keyed-Hash Message Authentication Code
IKE	Internet Key Exchange
J-PAKE	Password Authenticated Key Exchange by Juggling
KAS	Key Agreement Scheme
KAT	Known Answer Test
KBKDF	Key-based Key Derivation Function
KW	Key Wrap
KWP	Key Wrap with Padding
MAC	Message Authentication Code
MD2	Message Digest 2
MD5	Message Digest 5
NIST	National Institute of Science and Technology
PAA	Processor Algorithm Acceleration
PCT	Pair-wise Consistency Test

PBKDF2	Password-based Key Derivation Function v2
PKCS	Public-Key Cryptography Standards
PSS	Probabilistic Signature Scheme
RC2	Rivest Cipher 2
RC4	Rivest Cipher 4
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
S/MIME	Secure/Multipurpose Internet Mail Extensions
SSC	Shared Secret Computation
SSP	Sensitive Security Parameter
SSL	Secure Socket Layer
TLS	Transport Layer Security
XCBC	XOR Cipher Block Chaining

Appendix B. References

FIPS 140-3	FIPS PUB 140-3 - Security Requirements For Cryptographic Modules March 2019 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf
FIPS 140-3 IG	Implementation Guidance for FIPS PUB 140-3 and the Cryptographic Module Validation Program https://csrc.nist.gov/Projects/cryptographic-module-validation-program/fips-140-3-ig-announcements
FIPS 180-4	**Secure Hash Standard (SHS)** March 2012 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
FIPS 186-4	**Digital Signature Standard (DSS)** July 2013 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS 197	**Advanced Encryption Standard** November 2001 https://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
FIPS 198-1	**The Keyed Hash Message Authentication Code (HMAC)** July 2008 https://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
PKCS#1	**Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1** February 2003 https://www.ietf.org/rfc/rfc3447.txt
PKCS#5	**Password-Based Cryptography Specification Version 2.0** September 2000 https://www.ietf.org/rfc/rfc2898.txt
PKCS#7	**Cryptographic Message Syntax Version 1.5** March 1998 https://www.ietf.org/rfc/rfc2315.txt
PKCS#11	**Cryptographic Token Interface Base Specification Version 3.0** June 2020 https://docs.oasis-open.org/pkcs11/pkcs11-base/v3.0/pkcs11-base-v3.0.pdf
PKCS#12	**Personal Information Exchange Syntax v1.1** July 2014 https://www.ietf.org/rfc/rfc7292.txt
RFC 3526	**More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)** May 2003 https://www.ietf.org/rfc/rfc3526.txt
RFC 5288	**AES Galois Counter Mode (GCM) Cipher Suites for TLS** August 2008 https://www.ietf.org/rfc/rfc5288.txt
RFC 7919	**Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)** August 2016 https://www.ietf.org/rfc/rfc7919.txt
RFC 8446	**The Transport Layer Security (TLS) Protocol Version 1.3** August 2018 https://www.ietf.org/rfc/rfc8446.txt

SP 800-38A	**Recommendation for Block Cipher Modes of Operation Methods and Techniques** December 2001 https://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
SP 800-38A Addendum	**Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode** October 2010 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a-add.pdf
SP 800-38B	**Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication** May 2005 https://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf
SP 800-38D	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC November 2007 https://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf
SP 800-38F	**Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping** December 2012 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf
SP 800-52r2	**Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations** August 2019 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf
SP800-56Ar3	**Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography** April 2018 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf
SP 800-56Cr2	**Recommendation for Key-Derivation Methods in Key-Establishment Schemes** August 2020 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr2.pdf
SP 800-90Ar1	**Recommendation for Random Number Generation Using Deterministic Random Bit Generators** June 2015 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
SP 800-90B	**Recommendation for the Entropy Sources Used for Random Bit Generation** January 2018 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf
SP 800-108r1	**NIST Special Publication 800-108 - Recommendation for Key Derivation Using Pseudorandom Functions** August 2022 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-108r1.pdf
SP 800-132	**Recommendation for Password-Based Key Derivation - Part 1: Storage Applications** December 2010 https://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf
SP 800-133r2	**Recommendation for Cryptographic Key Generation** June 2020 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-133r2.pdf
SP 800-135r1	**Recommendation for Existing Application-Specific Key Derivation Functions** December 2011 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf
SP 800-140B	CMVP Security Policy Requirements March 2020 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-140B.pdf