SUSE Linux Enterprise Server NSS Cryptographic Module

version 3.0

FIPS 140-2 Non-Proprietary Security Policy

Doc version 3.0.5

Last update: 2021-11-23

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of contents

1 Cryptographic Module Specification

This document is the non-proprietary security policy for the SUSE Linux Enterprise Server NSS Cryptographic Module version 3.0. It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS 140-2 (Federal Information Processing Standards Publication 140-2) for a security level 1 module.

FIPS 140-2 details the requirements of the Governments of the U.S. and Canada for cryptographic modules, aimed at the objective of protecting sensitive but unclassified information. For more information on the FIPS 140-2 standard and validation program please refer to the NIST website at http://csrc.nist.gov/.

Throughout the document, “the NSS module” and “the module” are also used to refer to the SUSE Linux Enterprise Server NSS Cryptographic Module version 3.0.

1.1 Module Overview

The SUSE Linux Enterprise Server NSS Cryptographic Module (hereafter referred to as "the module") is a software library implementing general purpose cryptographic algorithms based on the industry standard PKCS#11 version 2.20. The module provides cryptographic services to applications running in the user space of the underlying operating system, through a C language, PKCS#11 compliant application program interface (API).

For the purpose of the FIPS 140-2 validation, the module is a software-only, multi-chip standalone cryptographic module validated at overall security level 1. Table 1 shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2 standard:

FIPS 140-2 Section		Security Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services and Authentication	2
4	Finite State Model	1
5	Physical Security	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self Tests	1
10	Design Assurance	2
11	Mitigation of Other Attacks	1

Table 1: Security Levels

Table 2 lists the software components of the cryptographic module, which defines its logical boundary. The module is provided for the 64-bit Intel architectures.

Component	Description
/usr/lib64/libsoftokn3.so	PKCS#11 wrapper shared library.
/usr/lib64/libsoftokn3.chk	DSA signature for libsoftokn3.so.
/usr/lib64/libnssdbm3.so	NSS database management shared library.
/usr/lib64/libnssdbm3.chk	DSA signature for libnssdbm3.so.
/lib64/libfreeblpriv3.so	General purpose cryptographic shared library.
/lib64/libfreeblpriv3.chk	DSA signature for libfreeblpriv3.so.

Table 2: Cryptographic Module Components

The software block diagram below shows the logical boundary of the module, and its interfaces with the operational environment.

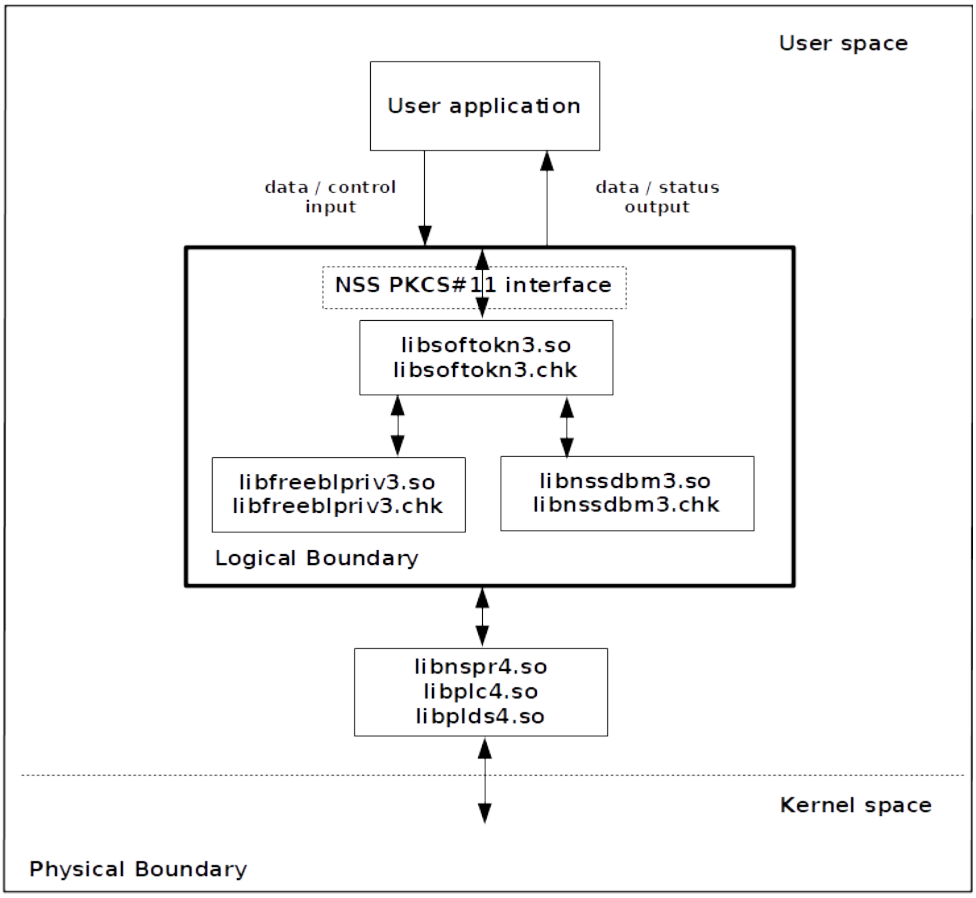

Figure 1: Software Block Diagram

Note: The libnspr4.so, libplc4.so and libplds4.so shared libraries are part of the mozilla-nspr package, which is a prerequisite for the module and part of the Operational Environment. See section 9.1.1 for installation instructions.

The module is aimed to run on a general purpose computer (GPC). Table 3 shows the platform on which the module has been tested:

Platform	Processor	Test Configuration
Dell EMC PowerEdge 640	Intel® Cascade Lake Xeon® Gold 6234	SUSE Linux Enterprise Server 15 SP0 with and without AES-NI (PAA).
Dell EMC PowerEdge 640	Intel® Cascade Lake Xeon® Gold 6234	SUSE Linux Enterprise Server 15 SP2 with and without AES-NI (PAA)
IBM System Z/15	IBM z15	SUSE Linux Enterprise Server 15 SP2
Gigabyte R181-T90	Cavium ThunderX2 CN9975 ARMv8	SUSE Linux Enterprise Server 15 SP2

Table 3: Tested Platforms

Note: Per FIPS 140-2 IG G.5, the Cryptographic Module Validation Program (CMVP) makes no statement as to the correct operation of the module or the security strengths of the generated keys when this module is ported and executed in an operational environment not listed on the validation certificate.

The physical boundary of the module is the surface of the case of the tested platform. Figure 2 shows the hardware block diagram including major hardware components of a GPC.

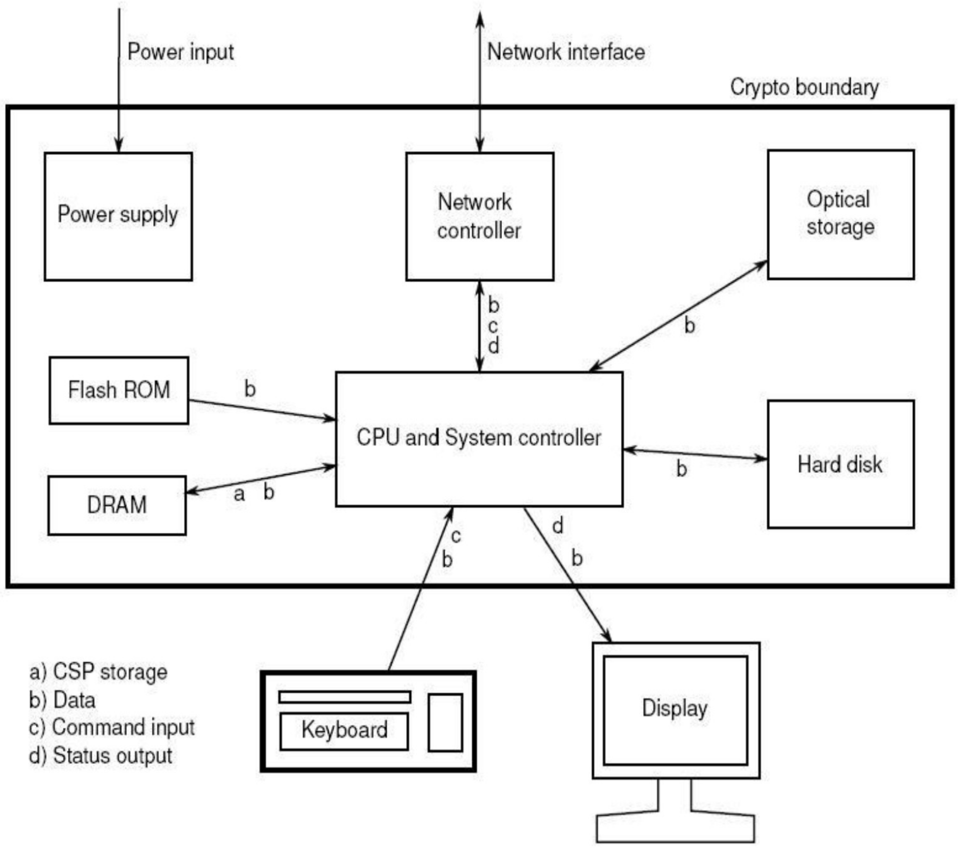

Figure 2: Hardware Block Diagram

1.2 Modes of Operation

The module supports two modes of operation:

- FIPS mode (the Approved mode of operation): only approved or allowed security functions with sufficient security strength can be used.
- non-FIPS mode (the non-Approved mode of operation): only non-approved security functions can be used.

The module enters FIPS mode after power-up tests succeed. Once the module is operational, the mode of operation is implicitly assumed depending on the security function invoked and the security strength of the cryptographic keys.

Critical security parameters (CSPs) used or stored in FIPS mode are not used in non-FIPS mode, and vice versa.

2 Cryptographic Module Ports and Interfaces

As a software-only module, the module does not have physical ports. For the purpose of the FIPS 140-2 validation, the physical ports are interpreted to be the physical ports of the hardware platform on which it runs.

The logical interfaces are the API through which applications request services. The ports and interfaces are shown in the following table.

FIPS Interface	Physical Port	Logical Interface
Data Input	None	API input parameters for data.
Data Output	None	API output parameters for data.
Control Input	None	API function calls, API input parameters for control input, /proc/sys/crypto/fips_enabled control file.
Status Output	None	API return codes, API output parameters for status output.
Power Input	PC Power Supply Port	N/A

Table 4: Ports and Interfaces

The module uses different function arguments for input and output to distinguish among 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. The module is designed with an input buffer that may hold security-related information, it zeroizes the buffer so that if the memory can be reused later as an output buffer. No sensitive information can be inadvertently leaked.

2.1 Inhibition of Data Output

All data output via the data output interface is inhibited when the module is performing power-up self-tests or is in the error state:

- During power-up self-tests: The module performs power-up self-tests automatically without any operator intervention. All data output via the data output interface is inhibited while self-tests are executed.
- In the error state: If the power-up self-tests fail, the module will be aborted and no service can be invoked. If the conditional self-tests fail during operation, the module will enter the error state and only the API functions that shut down and restart the module, reinitialize the module, or output status information can be invoked. These functions are `FC_GetFunctionList`, `FC_Initialize`, `FC_Finalize`, `FC_GetInfo`, `FC_GetSlotList`, `FC_GetSlotInfo`, `FC_GetTokenInfo`, `FC_InitToken`, `FC_CloseSession`, and `FC_CloseAllSessions`.

2.2 Output Data Path during key processing

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

3 Roles, Services and Authentication

3.1 Roles

The module supports the following roles:

- User role: performs all cryptographic services (in both FIPS mode and non-FIPS mode), including those that require authentication and access to secret or private keys.
- Crypto Officer role: performs module installation, configuration and initialization; and cryptographic services that do not require authentication, like message digest, random number generation, and status services.

3.2 Services

The module provides services via an application program interface (API) that is compliant with the PKCS#11 standard. The API functions are available to the calling application via the `FC_GetFunctionList` function, which in the only function exported and thus callable by its name. The rest of the API functions are accessible once this function returns a `CK_FUNCTION_LIST` structure containing the corresponding function pointers to the API functions.

This security policy uses the naming convention provided by the API documentation which defines the API functions prefixed with "`FC_`" (e.g. `FC_GetFunctionList`, `FC_Initialize`). Please refer to section 9 for how to initialize the module and invoke the API functions.

Services are available to users that assume one of the available roles. Crypto Officer role services do not require operator authentication, whereas user role services requires operator authentication, as they access secret and private keys and other CSPs associated with the user role.

For instance, message digest services are available to the Crypto Officer role only when CSPs are not accessed; the `FC_DigestKey` function computes the message digest (hash) of the value of a secret key, so it is available only to the User role. User role services, which access CSPs (e.g. `FC_GenerateKey`, `FC_GenerateKeyPair`), always require operator authentication.

Table 5 lists the services available in the module. FIPS-approved services must be requested using the FIPS-approved security functions specified in Table 6, or the FIPS-allowed security functions specified in Table 7. Invoking the services with those security functions implicitly turns the module into FIPS mode of operation.

Non-approved services are requested using the same API functions specified in Table 5, but using the non-approved security functions specified in Table 8. Invoking the services with those security functions implicitly turns the module into non-FIPS mode of operation.

For each service, the table lists the associated API functions, the role that can perform the service (User for the user role, CO for the Crypto Officer role), the cryptographic keys or CSPs involved, and their access type(s). The following convention is used to specify access rights to a CSP:

- *Create*: the calling application can create a new CSP.
- *Read*: the calling application can read the CSP.
- *Update*: the calling application can write a new value to the CSP.
- *Zeroize*: the calling application can zeroize the CSP.
- *n/a*: the calling application does not access any CSP or key during its operation.

Service	Role	API Function	Description	Keys/CSPs	Access
Get the list of API functions	CO User	FC_GetFunctionList	Return a list of function pointers.	None	n/a
Module Initialization	CO User	FC_InitToken	Initialize the token.	User Password All keys in key database	Zeroize
	CO User	FC_InitPIN	Set the initial user's password.	User Password	Create
General Purpose	CO User	FC_Initialize	Initialize the module library.	None	n/a
	CO User	FC_Finalize	Finalize (shutdown) the module library.	All keys	Zeroize
	CO User	FC_GetInfo	Obtain general information about the library.	None	n/a
Slot and Token Management	CO User	FC_GetSlotList	Obtain the list of slots in the system.	None	n/a
	CO User	FC_GetSlotInfo	Obtain information about a particular slot.	None	n/a
	CO User	FC_GetTokenInfo	Obtain information about the token.	None	n/a
	CO User	FC_GetMechanismList	Obtain the list of mechanisms (cryptographic algorithms) supported by the token	None	n/a
	CO User	FC_GetMechanismInfo	Obtain information about a particular mechanism.	None	n/a
	User	FC_SetPIN	Change the user's password.	User Password	Update
Session Management	CO User	FC_OpenSession	Open a connection between the application and a token.	None	n/a
	CO User	FC_CloseSession	Close a session.	All keys in session.	n/a
	CO User	FC_CloseAllSessions	Close all sessions in a token.	All keys in sessions.	Zeroize
	CO User	FC_GetSessionInfo	Obtain information about the session.	None	n/a
	CO User	FC_GetOperationState	Save the state of the session (only implemented for message digest).	None	n/a
	CO User	FC_SetOperationState	Restore the state of the session (only implemented for message digest).	None	n/a
	User	FC_Login	Log into a token.	User password	Read
	User	FC_Logout	Log out from a token.	None	n/a
Object	User	FC_CreateObject	Create a new object	None	n/a

Service	Role	API Function	Description	Keys/CSPs	Access
Management	User	FC_CopyObject	Create a copy of an object	Original key (any key type).	Read
				New key (same as original key).	Create
	User	FC_DestroyObject	Destroy an object	Any key type.	Zeroize
	User	FC_GetObjectSize	Obtain the size of an object.	Any key type.	Read
	User	FC_GetAttributeValue	Obtain an attribute value of an object.	Any key type.	Read
	User	FC_SetAttributeValue	Modify an attribute value of an object.	Any key type.	Update
	User	FC_FindObjectsInit	Initialize an object search operation.	None	n/a
	User	FC_FindObjects	Continue an object search operation	Any key type matching the search criteria.	Read
	User	FC_FindObjectsFinal	Finish an object search operation.	None	n/a
Data Encryption	User	FC_EncryptInit	Initialize encryption operation.	AES/Triple-DES key	Read
	User	FC_Encrypt	Single-part encryption.	AES/Triple-DES key	Read
	User	FC_EncryptUpdate	Continue multi-part encryption	AES/Triple-DES key	Read
	User	FC_EncryptFinal	Finish multi-part encryption.	AES/Triple-DES key	Read
Data Decryption	User	FC_DecryptInit	Initialize decryption operation.	AES/Triple-DES key	Read
	User	FC_Decrypt	Single-part decryption.	AES/Triple-DES key	Read
	User	FC_DecryptUpdate	Continue multi-part decryption	AES/Triple-DES key	Read
	User	FC_DecryptFinal	Finish multi-part decryption.	AES/Triple-DES key	Read
Message Digest	CO User	FC_DigestInit	Initialize message digest operation.	None	n/a
	CO User	FC_Digest	Single-part message digest.	None	n/a
	CO User	FC_DigestUpdate	Continue multi-part message digest.	None	n/a
	User	FC_DigestKey	Continue multi-part message digest using key.	HMAC key	Read
	CO User	FC_DigestFinal	Finish multi-part message digest.	None	n/a
Signature Generation, MAC generation	User	FC_SignInit	Initialize signature generation.	DSA/ECDSA/RSA private key, HMAC key	Read
	User	FC_Sign	Single-part signature generation.	DSA/ECDSA/RSA private key, HMAC key	Read

Service	Role	API Function	Description	Keys/CSPs	Access
	User	FC_SignUpdate	Continue multi-part signature generation.	DSA/ECDSA/RSA private key, HMAC key	Read
	User	FC_SignFinal	Finish multi-part signature generation.	DSA, ECDSA, RSA private keys, HMAC key	Read
	User	FC_SignRecoverInit	Initialize signature generation, where the data can be recovered from the signature.	DSA, ECDSA, RSA private keys, HMAC key	Read
	User	FC_SignRecover	Single-part signature generation where the data can be recovered from the signature.	DSA, ECDSA, RSA private keys, HMAC key	Read
Signature Verification, MAC Verification	User	FC_VerifyInit	Initialize signature verification.	DSA, ECDSA, RSA public keys, HMAC key	Read
	User	FC_Verify	Single-part signature verification.	DSA, ECDSA, RSA public keys, HMAC key	Read
	User	FC_VerifyUpdate	Continue multi-part signature verification.	DSA, ECDSA, RSA public keys, HMAC key	Read
	User	FC_VerifyFinal	Finish multi-part signature verification.	DSA, ECDSA, RSA public key, HMAC key	Read
	User	FC_VerifyRecoverInit	Initialize signature verification, where the data is recovered from the signature.	DSA, ECDSA, RSA public key, HMAC key	Read
	User	FC_VerifyRecover	Single-part signature verification where the data is recovered from the signature.	DSA, ECDSA, RSA public keys, HMAC key	Read
Dual-function Crypto Operations	User	FC_DigestEncryptUpdate	Continue a multi-part digesting and encryption operation.	AES, Triple-DES keys	Read
	User	FC_DecryptDigestUpdate	Continue a multi-part decryption and digesting operation.	AES, Triple-DES keys	Read
	User	FC_SignEncryptUpdate	Continue a multi-part signing and encryption operation.	DSA, ECDSA, RSA private keys, HMAC key	Read
	User	FC_DecryptVerifyUpdate	Continue a multi-part decryption and verifying operation.	DSA, ECDSA, RSA public keys, HMAC key	Read
Key Generation	User	FC_GenerateKey	Generate symmetric key.	AES, Triple-DES, HMAC keys	Create
			Generate TLS pre-master secret.	TLS pre-master secret	Create
	User	FC_GenerateKeyPair	Generate asymmetric key.	DSA, ECDSA, RSA key pairs	Create
Key Agreement	User	FC_GenerateKeyPair	Generate assymmetric key	Diffie-Hellman and EC Diffie-Hellman key pairs	Create

Service	Role	API Function	Description	Keys/CSPs	Access
	User	FC_DeriveKey	Shared secret computation	Diffie-Hellman and EC Diffie-Hellman key pairs	Read
				Diffie-Hellman and EC Diffie-Hellman shared secrets	Create
Key Transport	User	FC_WrapKey	Wrap and output a key using AES(KW) or RSA encapsulation.	Wrapping key	Read
				Key to wrap	Read
	User	FC_UnwrapKey	Wrap and import a key using AES(KW) or RSA encapsulation.	Wrapping key	Read
				Key to unwrap	Create
Key Derivation	User	FC_DeriveKey	Derive a key from TLS pre-master secret using TLS KDF.	TLS pre-master secret	Read
				TLS master secret	Create
			Derive a key from TLS master secret using TLS KDF.	TLS master secret	Read
				TLS derived keys	Create
	User	FC_DeriveKey	Derive keys from IKE shared secret using IKE KDF.	IKE shared secret	Read
				IKE derived keys	Create
	User	FC_GenerateKey	Derive a key from a password or passphrase using PBKDF	Password or passphrase	Read
				PBKDF derived key	Create
Random Number Generation	CO User	FC_SeedRandom	Provide additional seed material to the DRBG.	DRBG V and C values.	Update
	CO User	FC_GenerateRandom	Generate random data.	DRBG V and C values.	Update
Parallel Function Management	CO User	FC_GetFunctionStatus	Returns value 0x00000051 (legacy function)	None	n/a
	CO User	FC_CancelFunction	Returns value 0x00000051 (legacy function)	None	n/a
Show Status	CO User	FC_GetTokenInfo	Obtain information about the token.	None	n/a
	CO User	FC_GetSessionInfo	Obtain information about the session.	None	n/a
Self tests	CO User	None	Self-tests are performed automatically when loading the module.	DSA 2048-bit public key.	Read
Zeroization	User	FC_DestroyObject	Destroy an object	Key stored in key database.	Zeroize
	CO User	FC_InitToken	Initialize the token.	All keys stored in key database	Zeroize
	CO User	FC_Finalize	Finalize (shutdown) the module.	All keys in all sessions.	Zeroize
	CO User	FC_CloseSession	Close the session.	All keys in the session.	Zeroize
	CO User	FC_CloseAllSessions	Close all sessions in a token.	All keys in all sessions.	Zeroize

Service	Role	API Function	Description	Keys/CSPs	Access
Module installation and configuration	CO	None	n/a	None	n/a

Table 5: Services

Notes:

1. "Any key type", "original key" and "new key" are any AES, Triple-DES, HMAC key or any DSA, ECDSA, RSA public/private key pairs.
2. "wrapping key" corresponds to the AES key or RSA public/private key pair used to wrap or unwrap another key.
3. "key to wrap" is the key (of any type) that is wrapped by the "wrapping key" and output from the module.
4. "key to unwrap" is the key (of any type) that is unwrapped by the "wrapping key" and input to the module.
5. "derived key" is the key obtained by a key derivation function (TLS KDF, IKE KDF, and PBKDF).

3.3 Operator Authentication

The module implements role-based authentication. The module implements a password-based authentication for the user role; the crypto officer is assumed by default and no authentication is required.

To perform any security services with the user role, an operator must log into the module and complete the authentication procedure using the password, which is unique to the user role operator. This authentication provides access to the certificate and private key databases, needed by the module to performed those services. There is only one password to access the databases.

The password is passed to the module via the `FC_Login` function as one of its input arguments and will not be displayed. The return value of the function is the only feedback mechanism, which does not provide any information that could be used to guess or determine the password. The password is initialized by the Crypto Officer role as part of module initialization via the `FC_InitPIN` function and can be changed by the user role operator via the `FC_SetPIN` function.

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

Once a password has been established for the module, the user role is allowed to use the security services if and only if the user role is successfully authenticated to the module. Password establishment and authentication are required for the operation of the module.

3.3.1 Strength of the Authentication Mechanism

The module enforces the following requirements on the user password during password initialization or change.

- The password must be at least seven characters long.
- The password must consist of characters from three or more of the following five character classes:
  - digits (0-9). The last character of the password is not counted for this character class.

- ASCII lowercase letters (a-z).
- ASCII uppercase letters (A-Z). The first character of the password is not counted for this character class.
- ASCII non-alphanumeric characters (space and other ASCII special characters such as '$', '!')
- non-ASCII characters (Latin characters such as 'é', 'ß'; Greek characters such as 'Ω', 'θ'; other non-ASCII special characters such as '¿')

* To estimate the maximum probability of a successful random guess of the password, 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, and the probability to guess every character successfully is $(1/10)^5$. (1/26) . (1/26) = 1/67,600,000.

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

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

3.4 Algorithms

The module provides a generic C implementation of cryptographic algorithms, as well as an implementation using AESNI instructions for the AES cryptographic algorithm on the Intel x86 architecture. Table 6 lists the approved algorithms, the CAVP certificates, and other associated information of the cryptographic implementations in FIPS mode.

Algorithm	Mode / Method	Key Lengths, Curves (in bits)	Use	Standard	CAVP Certs
AES	ECB, CBC, CTR	128, 192, 256	Data encryption and decryption	FIPS197, SP800-38A	#A245 #A247 #A473 #A474
	KW	128, 192, 256	Key wrapping and unwrapping	SP800-38F	#A337 #A338 #A476 #A477
DRBG	Hash_DRBG: SHA-256 with and without PR	N/A	Deterministic random bit generation	SP800-90A	#A245 #A473
DSA		L=2048, N=224 L=2048, N=256 L=3072, N=256	Key pair generation	FIPS186-4	#A245 #A473
	SHA-224	L=2048, N=224	Domain parameter generation		
	SHA-256	L=2048, N=256 L=3072, N=256			
	SHA-224, SHA-256	L=2048, N=224 L=2048, N=256 L=3072, N=256	Digital signature generation		

Algorithm	Mode / Method	Key Lengths, Curves (in bits)	Use	Standard	CAVP Certs
	SHA-1	L=1024, N=160	Domain parameter verification		
	SHA-224	L=2048, N=224			
	SHA-256	L=2048, N=256 L=3072, N=256			
	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	L=1024, N=160 L=2048, N=224 L=2048, N=256 L=3072, N=256	Digital signature verification		
ECDSA		P-256, P-384, P-521	Key pair generation Public key verification	FIPS186-4	#A245 #A473
	SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Digital signature generation		
	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Digital signature verification		
HMAC	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message authentication code	FIPS198-1	#A245 #A473
KAS-ECC-SSC	ephemeralUnified scheme KAS Role: initiator, responder	P-256, P-384, P-521	EC Diffie-Hellman shared secret computation	SP800-56Ar3	#A681 #A682
KAS-FFC-SSC	dhEphem scheme KAS Role: initiator, responder	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Diffie-Hellman shared secret computation	SP800-56Ar3	#A681 #A682
PBKDF	HMAC with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512		Key derivation	SP800-132	#A245 #A473
KDF TLS	TLS v1.0/1.1, v1.2 with SHA-256, SHA-384, SHA-512		Key derivation	SP800-135	CVLs. #A245 #A473
KDF IKE	IKEv1 and IKEv2 with SHA-1, SHA-256, SHA-384, SHA-512		Key derivation	SP800-135	CVLs. #A246 #A475
RSA		2048, 3072, 4096	Key pair generation	FIPS186-4 (B.3.3)	#A245 #A473
	PKCS#1v1.5: SHA-256, SHA-384, SHA-512	2048, 3072, 4096	Digital signature generation	FIPS186-4	

Algorithm	Mode / Method	Key Lengths, Curves (in bits)	Use	Standard	CAVP Certs
	PKCS#1v1.5: SHA-1, SHA-256, SHA-384, SHA-512	2048, 3072, 4096	Digital signature verification		
Safe Primes Key Generation		Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Safe Primes Key Generation	SP800-56Ar3	#A681 #A682
SHS	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message digest	FIPS180-4	#A245 #A473
Triple-DES	ECB, CBC	192 (two-key Triple-DES)	Data decryption	SP800-67 SP800-38A	#A245 #A473
		192 (three-key Triple-DES)	Data encryption and decryption		
CKG			AES, Triple-DES, HMAC key generation	SP800-133	vendor affirmed
			RSA, DSA, ECDSA public and private key generation	FIPS186-4	#A245 #A473
KTS	AES KW	128, 192, 256	Key Wrapping and unwrapping	SP800-38F	#A337 #A338 #A476 #A477

Table 6: Approved Cryptographic Algorithms

3.5 Allowed Algorithms

Table 7 describes the non-approved but allowed algorithms in FIPS mode.

Algorithm	Use
NDRNG	The module obtains the entropy data from a NDRNG to seed the DRBG.
RSA key encapsulation with encryption and decryption primitives with keys equal or larger than 2048 bits up to 15360 or more.	Key establishment; allowed per [FIPS140-2_IG] D.9

Table 7: Non-Approved but Allowed Algorithms

3.6 Non-Approved Algorithms

Table 8 shows the non-Approved cryptographic algorithms implemented in the module that are only available in non-FIPS mode.

Algorithm	Use
AES in CTS mode	Data encryption and decryption
AES in GCM mode[1], CBC-MAC and XCBC-MAC modes	Authenticated data encryption and decryption
AES in KWP mode	Key wrapping
Camellia, CAST, CAST3, CAST5, ChaCha20, DES, IDEA, RC2, RC4, RC5, SEED	Key generation, data encryption and decryption
2-key Triple-DES	Key generation, data encryption
Triple-DES in CBC-MAC mode	Authenticated data encryption and decryption
Poly1305	Authenticated data encryption and decryption, message authentication code
MD2, MD5	Message digest
CMAC for AES	Message authentication code
HMAC using keys less than 112 bits of length HMAC with non-approved message digest algorithms	Message authentication code
GMAC	Message authentication code
SHA-1	Message digest in digital signature generation
DSA with L=1024 N=160	Key pair generation, domain parameter generation, digital signature generation
DSA with L=2048 N=224, L=2048 N=256 or L=3072 N=256 and using SHA-1, SHA-384, or SHA-512	Digital signature generation
RSA PSS	Digital signature generation and verification
RSA PKCS#1v1.5 with keys smaller than 2048 bits or greater than 4096 bits	Key pair generation, digital signature generation
RSA PKCS#1v1.5 with keys smaller than 1024 bits or greater than 4096 bits	Digital signature verification
RSA PKCS#1v1.5 with keys smaller than 2048 bits	Key encapsulation
Curve25519	Key pair generation, domain parameter generation and verification, digital signature generation and verification
J-PAKE	Key agreement
CDMF	Key generation, data encryption and decryption
HKDF, PBKDF1	Key derivation
Diffie-Hellman with keys generated with domain parameters other than safe primes.	Shared Secret computation.

1 AES in GCM mode does not meet IG A.5 requirements.

Algorithm	Use
EC Diffie-Hellman with P-192 curve, K curves, B curves and non-NIST curves.	Shared Secret computation.

Table 8: Non-Approved Cryptographic Algorithms

4 Physical Security

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

5 Operational Environment

This module operates in a modifiable operational environment per the FIPS 140-2 level 1 specifications. The module runs on a commercially available general-purpose operating system executing on the hardware specified in Table 3.

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

- SLES
- SLES for SAP
- SLED
- SLE Micro

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

Note: 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.

5.1 Policy

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

The application that requests cryptographic services is the single user of the module.

The ptrace system call, the debugger gdb and strace, as well as other tracing mechanisms offered by the Linux environment (ftrace, systemtap) shall not be used.

6 Cryptographic Key Management

Table 9 summarizes the Critical Security Parameters (CSPs) that are used by the cryptographic services implemented in the module. Key sizes allowed in the approved mode of operation are specified in Table 6 and Table 7.

Name	Generation	Entry and Output	Zeroization
AES keys	Generated using the SP800-90A DRBG via the FC_GenerateKey function.	Keys are input and output in encrypted form via FC_UnwrapKey and FC_WrapKey functions.	Zeroized in RAM with `FC_Finalize`, `FC_CloseSession` or `FC_CloseAllSession`. Zeroized in key storage with `FC_DestroyObject` or `FC_InitToken`.
Triple-DES keys			
HMAC keys			
RSA public and private keys	Public and private keys are generated using the FIPS 186-4 key generation method via the FC_GenerateKeyPair function; the key seed is obtained from the SP800-90A DRBG.	Keys are input and output in encrypted form via FC_UnwrapKey and FC_WrapKey functions.	
DSA public and private keys			
ECDSA public and private keys			
Diffie-Hellman public and private keys	Public and private keys are generated using the SP800-56ARev3 Safe Primes key generation method via the FC_GenerateKeyPair function; random values are obtained from the SP800-90A DRBG.		
EC Diffie-Hellman public and private keys	Public and private keys are generated using the FIPS 186-4 key generation method via the FC_GenerateKeyPair function; random values are obtained from the SP800-90A DRBG.		
Diffie-Hellman shared secret	Generated during shared secret computation via FC_DeriveKey.	Shared secret is output in plaintext form.	Zeroized in RAM with `FC_Finalize`, `FC_CloseSession` or `FC_CloseAllSession`. Zeroized in key storage with `FC_DestroyObject` or `FC_InitToken`.
EC Diffie-Hellman shared secret			
Password or passphrase	Not applicable. The password is entered via API parameters.	The password is passed into the module via API input parameters in plaintext.	Zeroized in RAM with `FC_Finalize`, `FC_CloseSession` or `FC_CloseAllSession`.
PBKDF derived key	Generated during the PBKDF	The key is output in encrypted form via FC_WrapKey functions.	Zeroized in key storage with `FC_DestroyObject` or `FC_InitToken`.
TLS pre-master secret	Generated using the SP800-90A DRBG via the FC_GenerateKey function. Generated during shared secret computation via FC_DeriveKey.	The key is output in encrypted form via FC_WrapKey functions.	Zeroized in RAM with `FC_Finalize`, `FC_CloseSession` or `FC_CloseAllSession`.
TLS master	Generated during the TLS	The TLS master secret is	Zeroized in key storage

Name	Generation	Entry and Output	Zeroization
secret	v1.0/1.1 and v1.2 KDFs from TLS pre-master secret.	output in plaintext form.	with FC_DestroyObject or FC_InitToken.
TLS derived keys	Generated during the TLS v1.0/1.1 and v1.2 KDFs from TLS master-secret	Keys are output in plaintext form.	Zeroized in key storage with FC_DestroyObject or FC_InitToken.
IKE shared secret	Obtained from Diffie-Hellman or EC Diffie-Hellman shared secret computation.	The IKE shared secret is output in plaintext form.	Zeroized in RAM with FC_Finalize, FC_CloseSession or FC_CloseAllSession.
IKE derived keys	Generated during the IKEv1 and IKEv2 KDFs	Keys are output in plaintext form.	Zeroized in key storage with FC_DestroyObject or FC_InitToken.
Entropy input string and seed material	Obtained from the NDRNG	Not applicable, it remains within the logical boundary.	Zeroized in RAM with FC_Finalize, FC_CloseSession or FC_CloseAllSession.
DRBG internal state: V and C values	Derived from entropy input as defined in SP800-90A	Not applicable, it remains within the logical boundary.	
User password for authentication	Not applicable; provided via API parameter.	User password is passed into the module via API input parameters in plaintext.	Zeroized in RAM with FC_Finalize, FC_CloseSession or FC_CloseAllSession.

Table 9: Life cycle of Keys or CSPs

The following sections describe how CSPs, in particular cryptographic keys, are managed during its life cycle.

6.1 Random Number Generation

The module employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90A] for the creation of seeds for symmetric keys, asymmetric keys, and DSA and ECDSA signature generation. In addition, the module provides a Random Number Generation service to calling applications.

The DRBG supports the Hash_DRBG mechanism using SHA-256 and without prediction resistance.

The module uses a Non-Deterministic Random Number Generator (NDRNG) as the entropy source for seeding the DRBG. The NDRNG is provided by the operational environment (i.e., Linux RNG), which is within the module’s physical boundary but outside of the module’s logical boundary. The NDRNG provides at least 256 bits of entropy to the DRBG during initialization (seed) and reseeding (reseed). The module periodically reseeds its DRBG: after 2^{48} calls to the random number generator, the module reseeds the DRBG automatically. The calling application can also enforce reseeding the DRBG by calling the FC_SeedRandom function.

The Linux kernel performs conditional self-tests on the output of NDRNG to ensure that consecutive random numbers do not repeat. The module performs the DRBG health tests as defined in section 11.3 of [SP800-90A].

6.2 Key/CSP Generation

The module provides an SP800-90A-compliant Deterministic Random Bit Generator (DRBG) for creation of symmetric keys, key components of asymmetric keys, and random number generation.

The key generation methods implemented in the module for Approved services in FIPS mode is compliant with [SP800-133] (vendor affirmed).

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 key agreement schemes are generated internally by the module using ECDSA key generation compliant with [FIPS186-4] and [SP800-56ARev3]. The Diffie-Hellman key agreement scheme is also compliant with [SP800-56ARev3], and generates keys using safe primes defined in RFC7919 and RFC3526, as described in the next section.

6.3 Key Agreement

The module provides Diffie-Hellman and EC Diffie-Hellman shared secret computation compliant with SP800-56ARev3, in accordance with scenario X1 (1) of IG D.8.

For Diffie-Hellman, the module supports the use of safe primes defined in RFC7919 for domain parameters and key generation, which are used in TLS key exchange. Note that the module only implements key generation and verification, 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 verification, 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 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.

6.4 Key Transport

The module provides the following key transport mechanisms:

- Key wrapping using AES-KW.

- RSA key encapsulation using private key encryption and public key decryption.

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

- AES key wrapping provides between 128 and 256 bits of encryption strength.
- RSA key wrapping[2] provides between 112 and 256 bits of encryption strength.

Note: As the module supports RSA key pairs greater than 2048 bits up to 15360 bits or more, the encryption strength 256 bits is claimed for RSA key encapsulation.

6.5 Key Derivation

The module supports the following key derivation methods according to [SP800-135]:

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

6.6 Key/CSP Entry and Output

The module does not support manual key entry or intermediate key generation key output. The keys are provided to the module via API input parameters in encrypted form (using the FC_UnwrapKey function) and output via API output parameters also in encrypted form (using the FC_WrapKey function).

6.7 Key/CSP Storage

The module employs the cryptographic keys and CSPs in FIPS Approved mode of operation as listed in Table 9. The module does not perform persistent storage of keys. Note that the private key database (provided with the files key3.db/key4.db) is within the module's physical boundary but outside its logical boundary.

6.8 Key/CSP Zeroization

The memory occupied by keys is allocated by regular memory allocation operating system calls. The application is responsible for calling the appropriate zeroization functions provided in the module's API and listed in Table 9:

- The `FC_Finalize`, `FC_CloseSession` or `FC_CloseAllSession` functions overwrite the memory occupied by keys with “zeros” and deallocate the memory with the regular memory deallocation operating system call.
- The `FC_DestroyObject` function overwrites with "zeros" the area occupied by the secret key in the private key database. The `FC_InitToken` function overwrites with "zeros" the whole private key database.

2 Key wrapping” is used instead of “key encapsulation” to show how the algorithm will appear in the certificate per IG G.13.

7 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The test platforms as shown in Table 3 are compliant to 47 CFR FCC Part 15, Subpart B, Class A (Business use).

8 Self Tests

8.1 Power-Up Tests

The module performs power-up tests when the module is loaded into memory, without operator intervention. Power-up tests ensure that the module is not corrupted and that the cryptographic algorithms work as expected.

While the module is executing the power-up tests, services are not available, and input and output are inhibited. The module is not available for use by the calling application until the power-up tests are completed successfully.

If any of the power-up test fails, the module enters the Error state. Subsequent calls to the module will also fail; no further cryptographic operations are possible. If the power-up tests complete successfully, the module will become operational and accept cryptographic operation service requests.

In order to verify whether the self-tests have succeeded, the calling application may invoke the `FC_Initialize` function. The function will return `CKR_OK` if the module is operational, `CKR_DEVICE_ERROR` if the module is in the Error state.

8.1.1 Integrity Tests

The integrity of the module is verified by performing a DSA signature verification for each component that comprises the module. The module uses DSA signature verification with a 2048-bit key and SHA-256. If the DSA signature for any of the components cannot be verified, the test fails and the module enters the error state.

8.1.2 Cryptographic Algorithm Tests

The module performs self-tests on all FIPS-Approved cryptographic algorithms supported in the Approved mode of operation, using the Known Answer Tests (KAT) shown in the following table.

Algorithm	Power-Up Tests
AES	KAT AES in ECB mode with 128, 192 and 256 bit keys, encryption and decryption (separately tested). KAT AES in CBC mode with 128, 192 and 256 bit keys, encryption and decryption (separately tested). KAT AES in KW mode with 128, 192 and 256 bit keys, encryption and decryption (separately tested).
Diffie-Hellman	Primitive “Z” Computation KAT with 2048-bit key
DRBG	KAT Hash_DRBG with SHA-256 without PR.
DSA	KAT DSA signature generation and verification with L=2048, N=224 and SHA-224 (separately tested).
EC Diffie-Hellman	Primitive “Z” Computation KAT with P-256 curve
ECDSA	KAT ECDSA signature generation and verification with P-256 and SHA-256 (separately tested).
HMAC	KAT HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512.
IKE KDF	SP800-135 IKE PRF using SHA-1, SHA-256, SHA-384 and SHA-512.
PBKDF KDF	KAT

Algorithm	Power-Up Tests
RSA	KAT RSA PKCS#1 v1.5 signature generation and verification with 2048-bit key and SHA-256, SHA-384 and SHA-512 (separately tested). KAT RSA with 2048-bit key, public key encryption and private key decryption (separately tested).
SHS	KAT SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512.
TLS KDF	TLS 1.0 PRF KAT and TLS 1.2 KAT using SHA-224, SHA-256, SHA-384 and SHA-512
Triple-DES	KAT Triple-DES ECB mode, encryption and decryption (separately tested). KAT Triple-DES CBC mode, encryption and decryption (separately tested).

Table 10: Self-Tests

For the KAT, the module calculates the result and compares it with the known value. If the answer does not match the known answer, the KAT fails and the module enters the Error state.

8.2 On-Demand Self-Tests

On-Demand self-tests can be invoked by powering-off and reloading the module, which cause the module to run the power-up tests again. During the execution of the on-demand self-tests, services are not available and no data output or input is possible.

During the execution of the on-demand self-tests, services are not available and no data output or input is possible.

8.3 Conditional Tests

The module performs conditional tests on the cryptographic algorithms, using the Pair-wise Consistency Tests (PCT) shown in the following table. If the conditional test fails, the module returns the `CKR_DEVICE_ERROR` error code to the calling application and enters the Error state. When the module is in the Error state, no data is output and cryptographic operations are not allowed.

Algorithm	Conditional Tests
DSA key generation	PCT using SHA-256, signature generation and verification.
ECDSA key generation	PCT using SHA-256, signature generation and verification.
RSA key generation	PCT using SHA-256, signature generation and verification. PCT using public encryption and private decryption.

Table 11: Conditional Tests

8.4 Error states

The Module enters the Error state returning the `CKR_DEVICE_ERROR` error code, on failure of power-on self-tests or conditional test. In the Error state, all data output is inhibited and no cryptographic operation is allowed. The error can be recovered by powering-off and reloading the module.

9 Guidance

9.1 Crypto Officer Guidance

The binaries of the module are contained in the RPM packages for delivery. The Crypto Officer shall follow this Security Policy to configure the operational environment and install the module to be operated as a FIPS 140-2 validated module.

The following RPM packages contain the FIPS validated module:

Processor Architecture	RPM Packages
Intel 64-bit	libsoftokn3-3.47.1-3.51.1.x86_64.rpm libsoftokn3-hmac-3.47.1-3.51.1.x86_64.rpm libfreebl3-3.47.1-3.51.1.x86_64.rpm libfreebl3-hmac-3.47.1-3.51.1.x86_64.rpm
IBM z15	libsoftokn3-3.47.1-3.51.1.s390x.rpm libsoftokn3-hmac-3.47.1-3.51.1.s390x.rpm libfreebl3-3.47.1-3.51.1.s390x.rpm libfreebl3-hmac-3.47.1-3.51.1.s390x.rpm
ARMv8 64-bit	libsoftokn3-3.47.1-3.51.1.aarch64.rpm libsoftokn3-hmac-3.47.1-3.51.1.aarch64.rpm libfreebl3-3.47.1-3.51.1.aarch64.rpm libfreebl3-hmac-3.47.1-3.51.1.aarch64.rpm

Table 12: RPM packages

9.1.1 Module Installation

The Netscape Portable Runtime (NSPR) package (mozilla-nspr-4.23-3.9.1.x86_64.rpm) is a prerequisite for the module. The mozilla-nspr package must be installed in the operating environment.

The Crypto Officer can install the RPM packages containing the module as listed in Table 12 using the zypper tool. The integrity of the RPM package is automatically verified during the installation, and the Crypto Officer shall not install the RPM package if there is any integrity error.

9.1.2 Operating Environment Configuration

The operating environment needs to be configured to support FIPS, so the following steps shall be performed with the root privilege:

1. Install the dracut-fips RPM package:

```
# zypper install dracut-fips
```

2. Recreate the INITRAMFS image:

```
# dracut -f
```

3. After regenerating the initrd, the Crypto Officer has to append the following parameter in the /etc/default/grub configuration file in the GRUB_CMDLINE_LINUX_DEFAULT line:

```
fips=1
```

4. After editing the configuration file, please run the following command to change the setting in the boot loader:

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

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

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

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

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

5. Reboot to apply these settings.

Now, the operating environment is configured to support FIPS operation. The Crypto Officer should check the existence of the file /proc/sys/crypto/fips_enabled, and verify it contains a numeric value “1”. If the file does not exist or does not contain “1”, the operating environment is not configured to support FIPS and the module will not operate as a FIPS validated module properly.

9.1.3 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/lib64/libaudit.so.1 to /usr/lib64/libaudit.so.1.0.0.

9.2 User Guidance

In order to run in FIPS mode, the module must be operated using the FIPS-approved services, with their corresponding FIPS-approved and FIPS-allowed cryptographic algorithms provided in this Security Policy (see section 3.2). In addition, key sizes must comply with [SP800-131A].

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

- Set the environment variable `NSS_ENABLE_AUDIT` to 1 before using the module.
- Use the `FC_GetFunctionList` function to obtain pointer references to the API. The function returns a `CK_FUNCTION_LIST` structure containing function pointers named as the API functions but with the "`C_`" prefix (e.g. `C_Initialize` and `C_Finalize`). The function pointers reference the "`FC_`" prefixed functions.
- Use `FC_Initialize` (function pointer `C_Initialize`) to initialize the module. Ensure that the function returns `CKR_OK`, which means that the module was properly configured and the power-on self-tests were successful. If the function returns a different code, the module must be reset and initialized again.

- For the first login, use FC_Login (function pointer C_Login) with a NULL password. This is required to set the initial user password of the token. Then, set the initial user role password using FC_InitPIN (function pointer C_InitPIN). Lastly, logout using the function FC_Logout (function pointer C_Logout).
- The user role can now be adopted on the module by logging in using the user password. The Crypto Officer role can be implicitly assumed by performing the Crypto Officer services as listed in Section 3.1.

The module can be configured to use different private key database formats: key3.db or key4.db. “key3.db” format is based on the Berkeley DataBase engine and should not be used by more than one process concurrently. “key4.db” format is based on SQL DataBase engine and can be used concurrently by multiple processes. Both databases are considered outside the cryptographic boundary and all data stored in these databases are considered stored in plaintext. The interface code of the NSS cryptographic 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 entered to the module from the calling application and output from the module to the calling application in encrypted form using the FC_WrapKey and FC_UnwrapKey functions, respectively. The cryptographic algorithms allowed for this purpose in the FIPS mode of operation are AES in KW mode, and RSA key encapsulation using the corresponding approved modes and key sizes.

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 Triple-DES encryption

Data encryption using the same three-key Triple-DES key shall not exceed 2^{16} Triple-DES blocks (2GB of data), in accordance to SP800-67 and IG A.13.

[SP800-67] imposes a restriction on the number of 64-bit block encryptions performed under the same three-key Triple-DES key.

When the three-key Triple-DES is generated as part of a recognized IETF protocol, the module is limited to 2^{20} 64-bit data block encryptions. This scenario occurs in the following protocols:

- Transport Layer Security (TLS) versions 1.1 and 1.2, conformant with [RFC5246]
- Secure Shell (SSH) protocol, conformant with [RFC4253]
- Internet Key Exchange (IKE) versions 1 and 2, conformant with [RFC7296]

In any other scenario, the module cannot perform more than 2^{16} 64-bit data block encryptions.

The user is responsible for ensuring the module’s compliance with this requirement.

9.2.2 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].

10 Mitigation of Other Attacks

10.1 Blinding Against RSA Timing Attacks

RSA is vulnerable to timing attacks. In a setup where attackers can measure the time of RSA decryption or signature operations, blinding must be used to protect the RSA operation from that attack.

The module uses the following blinding technique: instead of using the RSA decryption directly, a blinded value $y = x\ r^e \bmod n$ is decrypted and the unblinded value $x' = y'\ r^{-1} \bmod n$ returned. The blinding value r is a random value with the size of the modulus *n*.

10.2 Cache invariant modular exponentiation

Modular exponentiation used in DSA and RSA is vulnerable to cache-timing attacks. The module implements a variant of the modular exponentiation proposed by Colin Percival to defend against these attacks.

10.3 Double-checking RSA signatures

Arithmetic errors in RSA signatures might leak the private key. The module verifies the RSA signature generated after the cryptographic operation is performed.

Appendix A - CAVP certificates

The tables below show the certificates obtained from the CAVP for all the target platforms included in Table 3. The CAVP certificates validate all algorithm implementations used as approved or allowed security functions in FIPS mode of operation. The tables include the certificate number, the label used in the CAVP certificate for reference and a description of the algorithm implementation.

Cert#	CAVP Label	Algorithm Implementation
A247	AESNI	AES using AESNI instructions.
A337	AESNI_KW	AES-KW using AESNI instructions
A246	IKE_KDF	Internet Key exchange key derivation function implementation.
A245	Generic C	Generic C implementation of cryptographic algorithms
A338	Generic C KW	Generic C implementation for key wrapping.
A681	SP800 56A rev 3	SP800-56A rev 3 compliant implementation.

Table 13: CAVP certificates for the Intel Xeon processor for SLES 15 SP0

Cert#	CAVP Label	Algorithm Implementation
A474	AESNI	AES using AESNI instructions.
A477	AESNI_KW	AES-KW using AESNI instructions
A475	IKE_KDF	Internet Key exchange key derivation function implementation.
A473	Generic C	Generic C implementation of cryptographic algorithms
A476	Generic C KW	Generic C implementation for key wrapping.
A682	SP800 56A rev 3	SP800-56A rev 3 compliant implementation.

Table 14: CAVP certificates for the Intel Xeon processor for SLES 15 SP2

Cert#	CAVP Label	Algorithm Implementation
A475	IKE_KDF	Internet Key exchange key derivation function implementation.
A473	Generic C	Generic C implementation of cryptographic algorithms
A476	Generic C KW	Generic C implementation for key wrapping.
A682	SP800 56A rev 3	SP800-56A rev 3 compliant implementation.

Table 15: CAVP certificates for the IBM z15 processor for SLES 15 SP2

Cert#	CAVP Label	Algorithm Implementation
A475	IKE_KDF	Internet Key exchange key derivation function implementation.
A473	Generic C	Generic C implementation of cryptographic algorithms
A476	Generic C KW	Generic C implementation for key wrapping.
A682	SP800 56A rev 3	SP800-56A rev 3 compliant implementation.

Table 16: CAVP certificates for the ARMv8 processor for SLES 15 SP2

Appendix B - Glossary and Abbreviations

AES	Advanced Encryption Specification
AES_NI	Intel® Advanced Encryption Standard (AES) New Instructions
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CCM	Counter with Cipher Block Chaining Message Authentication Code
CDMF	Commercial Data Masking Facility
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
FIPS	Federal Information Processing Standards Publication
GCM	Galois Counter Mode
HKDF	HMAC-based Extract-and-Expand Key Derivation Function
HMAC	Hash Message Authentication Code
IDEA	International Data Encryption Algorithm
J-PAKE	Password Authenticated Key Exchange by Juggling
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
PKCS	Public Key Cryptography Standards
RNG	Random Number Generator
RPM	Red hat Package Manager
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
TDES	Triple-DES

Appendix C - References

FIPS 140-2	**FIPS PUB 140-2 - Security Requirements for Cryptographic Modules** http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
FIPS 140-2_IG	**Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program** December 3, 2019 http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf
FIPS180-4	**Secure Hash Standard (SHS)** http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
FIPS186-4	**Digital Signature Standard (DSS)** http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS197	**Advanced Encryption Standard** http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
FIPS198-1	**The Keyed Hash Message Authentication Code (HMAC)** http://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** http://www.ietf.org/rfc/rfc3447.txt
SP800-38A	**NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques** http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf
SP800-38B	**NIST Special Publication 800-38B - Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication** http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf
SP800-38D	**NIST Special Publication 800-38D - Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC** http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf
SP800-38F	**NIST Special Publication 800-38F - Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping** http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf
SP800-67	**NIST Special Publication 800-67 Revision 1 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher** http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-67r1.pdf
SP800-90A	**NIST Special Publication 800-90A Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

SP800-131A **NIST Special Publication 800-131A Revision 1- Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths**
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

SP800-132 **NIST Special Publication 800-132 - Recommendation for Password-Based Key Derivation - Part 1: Storage Applications**
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf