SUSE Linux Enterprise Server 12 - NSS Module v1.0

FIPS 140-2 Non-Proprietary Security Policy

Version 1.5

Last Update: 2016-04-20

Prepared by:
atsec information security corporation
9130 Jollyville Road, Suite 260
Austin, TX 78759
www.atsec.com

Contents

1 Cryptographic Module Specification

This document is the non-proprietary security policy for the SUSE Linux Enterprise Server 12 - NSS Module, and was prepared as part of the requirements for conformance to Federal Information Processing Standard (FIPS) 140-2, Security Level 2.

1.1 Description of the Module

The SUSE Linux Enterprise Server 12 - NSS Module (hereafter referred to as the “Module”) is a software library supporting FIPS 140-2 Approved cryptographic algorithms. The current version of the Module is 1.0. The Module is an open-source, general-purpose cryptographic library, with an API based on the industry standard PKCS #11 version 2.20. For the purposes of FIPS 140-2 validation, the Module is classified as a software-only module. Its embodiment type is defined as multi-chip standalone.

The Module is FIPS140-2 validated at overall Security Level 2 with levels for individual sections shown in the table below:

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

Table 1: Security Level of the Module

The Module has been tested on the following platform:

Manufacturer	Model	O/S & Ver.
HP	ProLiant DL320e Gen8	SUSE Linux Enterprise Server 12

Table 2: Tested Platform

The Module has been tested in the following configuration:

- 64-bit x86_64 without AES-NI

Note: The test platform supports AES-NI instruction set, but it is disabled in the Module. So the Module always uses C implementation of AES.

1.2 Description of Approved Modes

The Module supports two modes of operation: FIPS Approved mode and non-Approved mode. When the Module is powered on, the power-up self-tests are executed automatically without any operator intervention.

If the power-up self-tests complete successfully, the Module will be in FIPS Approved mode. Table 3 lists the services using Approved algorithms in FIPS Approved mode.

Service	Algorithm	Keys/CSPs	CAVS Certificate
Encryption and decryption	C implementation of AES • ECB • CBC • CTR • GCM	AES 128, 192 and 256 bits keys	Cert. #3452
	Triple-DES • ECB • CBC • CTR	Triple-DES 168 bits keys	Cert. #1943
Signature generation and verification	DSA domain parameter generation, key pair generation and signature generation	DSA 2048 and 3072 bits keys	Cert. #971
	DSA domain parameter verification and signature verification	DSA 1024, 2048 and 3072 bits keys	
	ECDSA key pair generation, public key verification, signature generation and signature verification	ECDSA keys with P-256, P-384 and P-521	Cert. #699
	RSA key pair generation and PKCS#1 v1.5 signature generation	RSA 2048 and 3072 bits keys	Cert. #1767
	RSA PKCS#1 v1.5 signature verification	RSA 1024, 2048 and 3072 bits keys	
Message digest	SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512	N/A	Cert. #2848
	HMAC • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512	At least 112 bits HMAC keys	Cert. #2198
Random number generation	SP800-90A Hash_based DRBG • SHA-256	Entropy input string, seed, V and C	Cert. #846

Table 3: Services using Approved Algorithms in FIPS Approved mode

Table 4 lists the services using non-Approved but allowed algorithms in FIPS Approved mode.

Service	Algorithm	Note	Keys/CSPs
Key management	AES key wrapping using approved mode of AES	AES with ECB, CBC and CTR mode is validated with CAVS cert. 3452	AES 128, 192 and 256 bits keys
	Triple-DES key wrapping using approved mode of Tripe-DES	Triple-DES with ECB, CBC and CTR mode is validated with CAVS cert. 1943	Triple-DES 168 bits keys
	RSA key wrapping (encrypt, decrypt)	The CAVP testing is not available	RSA keys with size equal to or larger than 2048 bits
	Diffie-Hellman key agreement	Not validated by CAVP	Diffie-Hellman with keys between 2048 and 15360 bits
	EC Diffie-Hellman key agreement	Not validated by CAVP	EC Diffie-Hellman private and public components with curves P-256, P-384 and P-521

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

Notes:

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

Caveat:

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

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

Service	Algorithm
Encryption and decryption	Camellia
	DES
	RC2
	RC4
	RC5
	SEED
	AES CTS block chaining mode

Service	Algorithm
Signature generation and verification	DSA domain parameter generation, key pair generation and signature generation with key size not equal to 2048 or 3072 bits
	DSA domain parameter verification and signature verification with key size not equal to 1024, 2048 or 3072 bits
	RSA key generation and PKCS#1 v1.5 signature generation with key size not equal to 2048 or 3072 bits
	RSA PKCS#1 v1.5 signature verification with key size not equal to 1024, 2048 or 3072 bits
	RSA PSS signature generation and verification
Message digest	MD2
	MD5
Key management	AES key wrapping based on SP800-38F but not validated by CAVP
	RSA key wrapping (encrypt, decrypt) with key size smaller than 2048 bits
	Diffie-Hellman key agreement with keys smaller than 2048 bits
	JPAKE key agreement

Table 5: Services using non-Approved Algorithms in non-Approved mode

1.3 Cryptographic Boundary

The Module's physical boundary is the surface of the case of the platform (depicted in the hardware block diagram).

The Module's logical boundary consists of the shared library files and their integrity check signature files, which are delivered with the RPM packages as listed below:

- The libsoftokn3-3.19.2_CKBI_1.98-21.1.x86_64.rpm, which contains the following shared libraries:

  /usr/lib64/libnssdbm3.so

  /usr/lib64/libsoftokn3.so
- The libsoftokn3-hmac-3.19.2_CKBI_1.98-21.1.x86_64.rpm, which contains the following integrity check signature files:

  /usr/lib64/libnssdbm3.chk

  /usr/lib64/libsoftokn3.chk
- The libfreebl3-3.19.2_CKBI_1.98-21.1.x86_64.rpm, which contains the following shared library:

  /lib64/libfreebl3.so
- The libfreebl3-hmac-3.19.2_CKBI_1.98-21.1.x86_64.rpm, which contains the following integrity check signature file:

  /lib64/libfreebl3.chk

The Module shall be installed and instantiated by the dracut-fips package with the RPM file version 037-37.2. The dracut-fips RPM package is only used for the configuration of the Module in every boot. This code is not active when the Module is operational and does not provide any services to users interacting with the Module. Therefore, the dracut-fips package is outside the Module's logical boundary.

1.3.1 Hardware Block Diagram

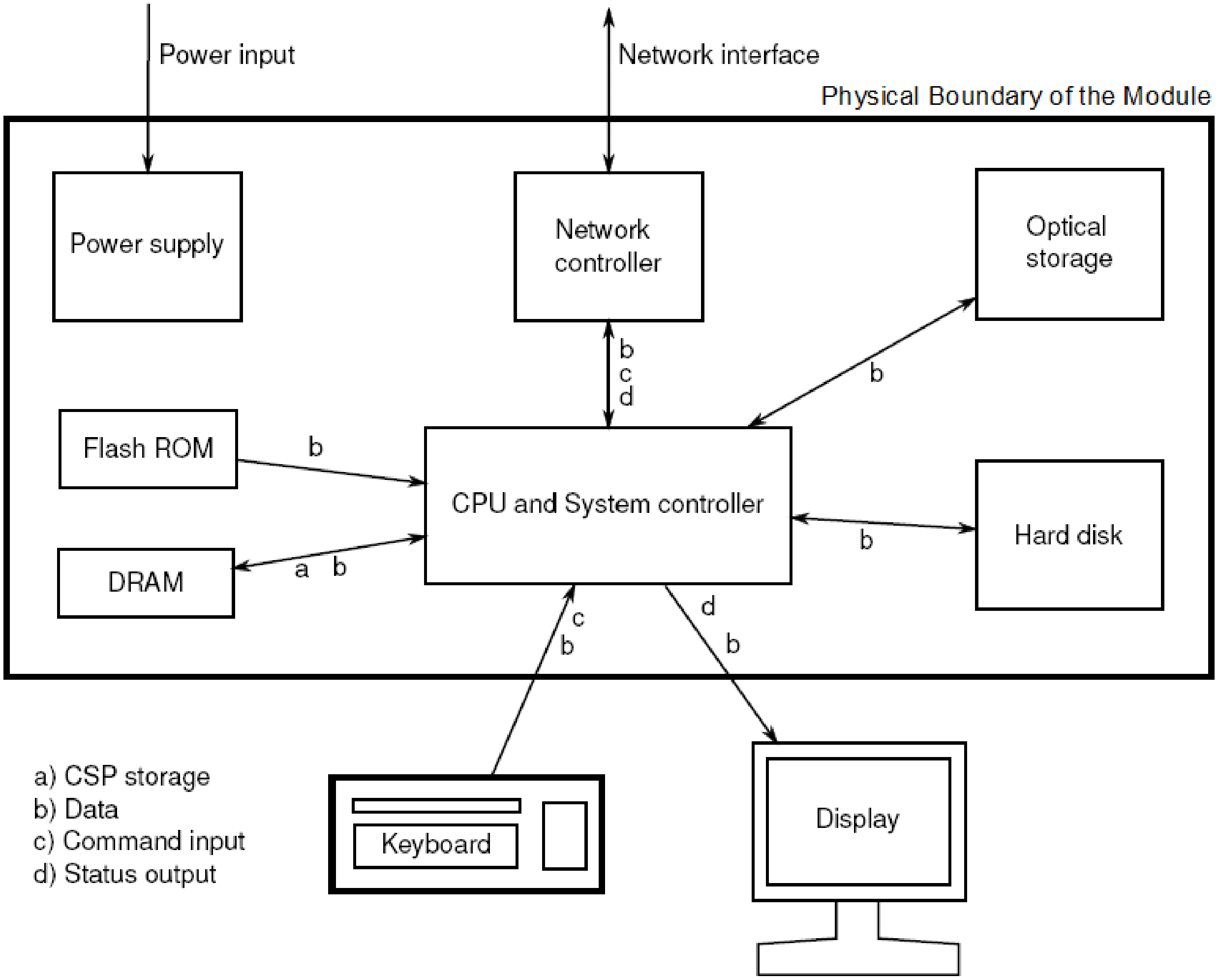

Figure 1. Hardware Block Diagram

1.3.2 Software Block Diagram

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

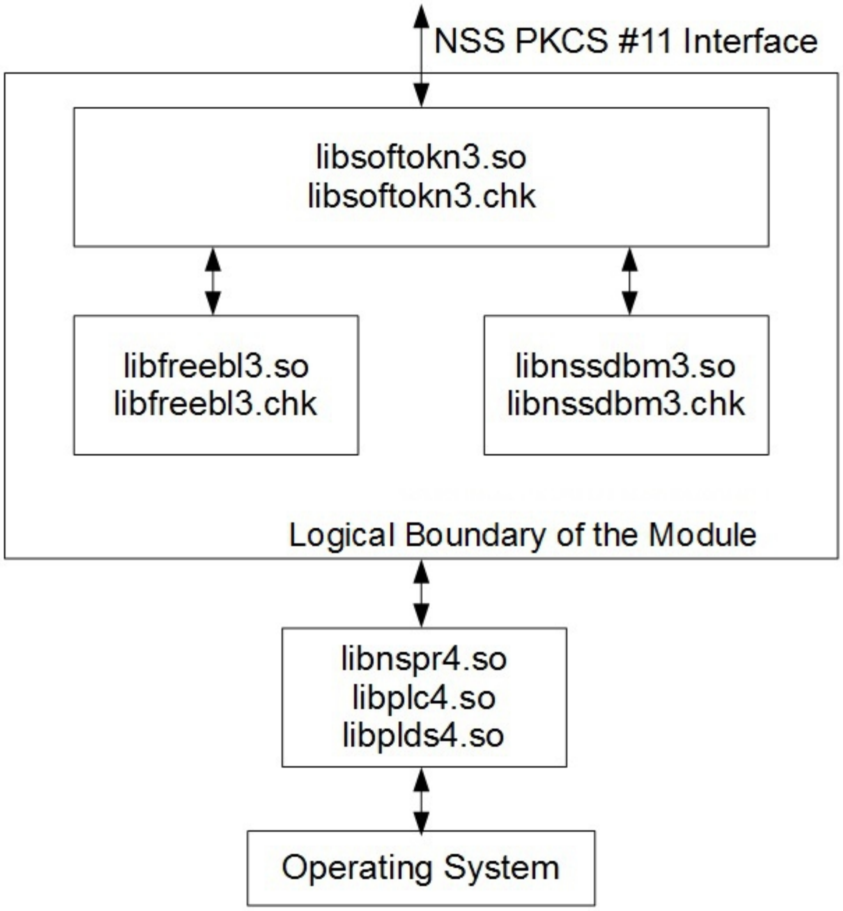

Figure 2. Software Block Diagram

2 Cryptographic Module Ports and Interfaces

The physical ports of the Module are the same as the computer system on which it is executing. The logical interface is a C-language Application Program Interface (API) following the PKCS #11 specification.

The Data Input interface consists of the input parameters of the API functions. The Data Output interface consists of the output parameters of the API functions. The Control Input interface consists of the actual API functions. The Status Output interface includes the return values of the API functions. The ports and interfaces are shown in the following table.

FIPS Interface	Physical Port	Module Interface
Data Input	N/A	API input parameters
Data Output	N/A	API output parameters
Control Input	N/A	API function calls, configuration file /proc/sys/crypto/fips_enabled
Status Output	N/A	API return codes and status parameters
Power Input	PC Power Supply Port	N/A

Table 6: 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. After the Module is done with an input buffer that holds security-related information, it always zeroizes the buffer so that if the memory is 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 in error states.

- 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 error states: 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 operational 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`, `FC_CloseAllSessions`, and `FC_WaitForSlotEvent`.

2.2 Disconnecting the Output Data Path from the Key Processes

During key generation and key zeroization, the Module may perform audit logging, but the audit records do not contain any sensitive information. The Module does not return any function output arguments 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

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

3.1 Roles

The Module implements two roles: User role and Crypto Officer (CO) role, their allowed services are listed in the following table.

Role	Descriptions
User	Perform general security services which use the secret or private keys of the Module. It is also responsible for the retrieval, updating, and deletion of keys from the private key database.
CO	Perform module installation, configuration and initialization. The CO role can access other general-purpose services (such as message digest and random number generation services) and status services of the Module. The CO does not have access to any service that utilizes the secret or private keys of the Module. The CO must control the access to the Module before and after installation, including management of physical access to the computer, execution of the Module, as well as management of the security facilities provided by the operating system.

Table 7: Roles

3.2 Role Assumption

The CO role is implicitly assumed by an operator while installing the Module by following the instruction in Section 9.1 and while performing other services as listed in Table 7.

The Module also implements a password-based authentication for the User role. To perform any security services under the User role, an operator must log into the Module and complete an authentication procedure using the password information unique to the User role operator. The password is passed to the Module via the API function as one of its input arguments and won't be displayed. The return value of the function is the only feedback mechanism, which does not provide any information that could be used to guess or determine the password. The password is initialized by the CO role as part of module initialization and can be changed by the User role operator.

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

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

3.3 Strength of Authentication Mechanism

In the FIPS Approved mode of operation, the Module imposes the following requirements on the password. These requirements are enforced by the Module on password initialization or change.

- The password must be at least seven characters long.
- The password must consist of characters from three or more character classes. We define five character classes: digits (0-9), ASCII lowercase letters (a-z), ASCII uppercase letters (A-Z), ASCII non-alphanumeric characters (space and other ASCII special characters such as '$', '!'), and non-ASCII

characters (Latin characters such as 'é', 'ß'; Greek characters such as 'Ω', 'θ'; other non-ASCII special characters such as '¿'). If an ASCII uppercase letter is the first character of the password, the uppercase letter is not counted toward its character class. Similarly, if a digit is the last character of the password, the digit is not counted toward its character class.

To estimate the maximum probability 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 equals to 1/67,600,000, which is smaller than the required threshold 1/1,000,000.

After each failed authentication attempt in the FIPS Approved 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 equals to 60 * (1/67,600,000) = 0.089 * (1/100,000), which is smaller than the required threshold 1/100,000.

3.4 Multiple Concurrent operators

The Module doesn't allow concurrent operators.

- On a multi-user operating system, this is enforced by making the NSS certificate and private key databases readable and writable by the owner of the files only.

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

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

3.5 Services

3.5.1 Calling Convention of API Functions

The Module has a set of API functions denoted by FC_xxx. All the API functions for the FIPS Approved mode of operation are listed in Table 8 of Section 3.5.2.

Among the Module's API functions, only FC_GetFunctionList is exported and therefore callable by its name. All the other API functions must be called via the function pointers returned by FC_GetFunctionList. It returns a CK_FUNCTION_LIST structure containing function pointers named C_xxx such as C_Initialize and C_Finalize. The C_xxx function pointers in the CK_FUNCTION_LIST structure returned by FC_GetFunctionList point to the FC_xxx functions.

The following convention is used to describe API function calls. Here FC_Initialize is used as an example:

- When “call FC_Initialize” is mentioned, the technical equivalent of “call the FC_Initialize function via the C_Initialize function pointer in the CK_FUNCTION_LIST structure returned by FC_GetFunctionList” is implied.

3.5.2 API Functions

The Module supports Crypto-Officer services which require no operator authentication, and User services which require operator authentication. Crypto-Officer services do not require access to the secret and private keys and other CSPs associated with the user. The message digesting services are available to Crypto-Officer only when CSPs are not accessed. User services which access CSPs (e.g., FC_GenerateKey, FC_GenerateKeyPair) require operator authentication.

Table 8 lists all the services available in FIPS Approved mode. Access types R, W and Z stand for Read, Write and Zeroize, respectively. Role types U and CO correspond to User role and Crypto Officer role, respectively. Please refer to Table 3 and Table 4 for the Approved or allowed key size of each cryptographic algorithm supported by the Module.

Note: The message digesting API functions (except FC_DigestKey) that do not use any keys of the Module are accessed to the Crypto-Officer role and do not require User role authentication to the Module. The FC_DigestKey API function computes the message digest (hash) of the value of a secret key, so it is available only to the User role.

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

Service	Role	API Function	Description	CSPs	Access
management			between an application and a particular token		
	CO	FC_CloseSession	Close a session	All keys for the session	Z
	CO	FC_CloseAllSessions	Close all sessions with a token	All keys	Z
	CO	FC_GetSessionInfo	Obtain information about the session. This function provides the Show Status service.	none	-
	CO	FC_GetOperationState	Save the state of the cryptographic operation in a session. This function is only implemented for message digest operations.	none	-
	CO	FC_SetOperationState	Restore the state of the cryptographic operation in a session. This function is only implemented for message digest operations.	none	-
	U	FC_Login	Log into a token	User password	R
	U	FC_Logout	Log out from a token	none	-
Object management	U	FC_CreateObject	Create a new object	key	W
	U	FC_CopyObject	Create a copy of an object	Original key	R
				New key	W
	U	FC_DestroyObject	Destroy an object	key	Z
	U	FC_GetObjectSize	Obtain the size of an object in bytes	key	R
	U	FC_GetAttributeValue	Obtain an attribute value of an object	key	R
	U	FC_SetAttributeValue	Modify an attribute value of an object	key	W
	U	FC_FindObjectsInit	Initialize an object search operation	none	-
	U	FC_FindObjects	Continue an object search operation	Keys matching the search criteria	R
	U	FC_FindObjectsFinal	Finish an object search operation	none	-
Encryption and decryption	U	FC_EncryptInit	Initialize an encryption operation	AES/Triple-DES secret key	R
	U	FC_Encrypt	Encrypt single-part data	AES/Triple-DES secret key	R

Service	Role	API Function	Description	CSPs	Access
	U	FC_EncryptUpdate	Continue a multiple-part encryption operation	AES/Triple-DES secret key	R
	U	FC_EncryptFinal	Finish a multiple-part encryption operation	AES/Triple-DES secret key	R
	U	FC_DecryptInit	Initialize a decryption operation	AES/Triple-DES secret key	R
	U	FC_Decrypt	Decrypt single-part encrypted data	AES/Triple-DES secret key	R
	U	FC_DecryptUpdate	Continue a multiple-part decryption operation	AES/Triple-DES secret key	R
	U	FC_DecryptFinal	Finish a multiple-part decryption operation	AES/Triple-DES secret key	R
Message digest	CO	FC_DigestInit	Initialize a message-digesting operation	none	-
	CO	FC_Digest	Digest single-part data	none	-
	CO	FC_DigestUpdate	Continue a multiple-part digesting operation	none	-
	U	FC_DigestKey	Continue a multi-part message-digesting operation by digesting the value of a secret key as part of the data already digested	HMAC key	R
	CO	FC_DigestFinal	Finish a multiple-part digesting operation	none	-
Signature generation and verification	U	FC_SignInit	Initialize a signature operation	DSA/ECDSA/ RSA private key, HMAC key	R
	U	FC_Sign	Sign single-part data	DSA/ECDSA/ RSA private key, HMAC key	R
	U	FC_SignUpdate	Continue a multiple-part signature operation	DSA/ECDSA/ RSA private key, HMAC key	R
	U	FC_SignFinal	Finish a multiple-part signature operation	DSA/ECDSA/ RSA private key, HMAC key	R
	U	FC_SignRecoverInit	Initialize a signature operation, where the data can be recovered from the signature	DSA/ECDSA/ RSA private key	R
	U	FC_SignRecover	Sign single-part data, where the data can be recovered	DSA/ECDSA/ RSA private key	R

Service	Role	API Function	Description	CSPs	Access
			from the signature		
	U	FC_VerifyInit	Initialize a verification operation	DSA/ECDSA/ RSA public key, HMAC key	R
	U	FC_Verify	Verify a signature on single-part data	DSA/ECDSA/ RSA public key, HMAC key	R
	U	FC_VerifyUpdate	Continue a multiple-part verification operation	DSA/ECDSA/ RSA public key, HMAC key	R
	U	FC_VerifyFinal	Finish a multiple-part verification operation	DSA/ECDSA/ RSA public key, HMAC key	R
	U	FC_VerifyRecoverInit	Initialize a verification operation where the data is recovered from the signature	DSA/ECDSA/ RSA public key	R
	U	FC_VerifyRecover	Verify a signature on single-part data, where the data is recovered from the signature	DSA/ECDSA/ RSA public key	R
Dual-function cryptographic operations	U	FC_DigestEncryptUpdate	Continue a multiple-part digesting and encryption operation	AES/Triple-DES secret key	R
	U	FC_DecryptDigestUpdate	Continue a multiple-part decryption and digesting operation	AES/Triple-DES secret key	R
	U	FC_SignEncryptUpdate	Continue a multiple-part signing and encryption operation	DSA/ECDSA/ RSA private key, HMAC key	R
				AES/Triple-DES secret key	R
	U	FC_DecryptVerifyUpdate	Continue a multiple-part decryption and verify operation	DSA/ECDSA/ RSA public key, HMAC key	R
				AES/Triple-DES secret key	R
Key management	U	FC_GenerateKey	Generate a secret key	AES/Triple-DES secret key	W
	U	FC_GenerateKeyPair	Generate a public/private key pair. This function performs the pair-wise consistency tests.	DSA/ECDSA/ RSA key pair, Diffie-Hellman/EC Diffie-Hellman public and private	W

Service	Role	API Function	Description	CSPs	Access
				components	
	U	FC_WrapKey	Wrap (encrypt) a key using one of the following mechanisms allowed in FIPS mode through December 31st 2017 per IG D.9: (1) RSA encryption (2) AES encryption (3) Triple-DES encryption	Wrapping key	R
				Key to be wrapped	R
	U	FC_UnwrapKey	Unwrap (decrypt) a key using one of the following mechanisms allowed in FIPS mode through December 31st 2017 per IG D.9: (1) RSA decryption (2) AES decryption (3) Triple-DES decryption	Unwrapping key	R
				Unwrapped key	W
	U	FC_DeriveKey	Derive a key from a base key	Base key	R
				Derived key	W
Random number generation	CO	FC_SeedRandom	Mix in additional seed material to the random number generator	Entropy string, seed, DRBG V and C values	RW
	CO	FC_GenerateRandom	Generate random data. This function performs the continuous random number generator test	Random data, DRBG V and C values	RW
Parallel function management	CO	FC_GetFunctionStatus	A legacy function, which simply returns the value 0x00000051 (function not parallel)	none	-
	CO	FC_CancelFunction	A legacy function, which simply returns the value 0x00000051 (function not parallel)	none	-
Self tests	CO	N/A	The self tests are performed automatically when loading the module	DSA 2048-bit public key	R
Zeroization	U	FC_DestroyObject	All CSPs are automatically zeroized when freeing the cipher handle	All secret or private keys and password	Z
	CO	FC_InitToken FC_Finalize FC_CloseSession FC_CloseAllSessions			

Table 8: Service Details

NOTE:

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

Please refer to Table 5 for the non-Approved services, and invocation of any of these services will put the module in non-Approved mode implicitly.

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 definition.

The underlying operating system, SUSE Linux Enterprise Server 12, is evaluated according to Common Criteria at EAL4 – certification ID of BSI-DSZ-CC-0962-2016 claiming compliance to the OSPP.

5.1 Policy

The operating system is restricted to a single operator mode of operation (i.e., concurrent operators are explicitly excluded).

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

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

6 Cryptographic Key Management

The management of all keys/CSPs used by the Module is summarized in the table below.

Key/CSP	Generation	Storage	Entry/Output	Zeroization
AES or Triple-DES key	SP 800-90A DRBG	Application memory or key database	Encrypted through key wrapping using FC_WrapKey	Automatically zeroized when freeing the cipher handle
DSA, ECDSA or RSA private key	SP 800-90A DRBG	Application memory or key database	Encrypted through key wrapping using FC_WrapKey	Automatically zeroized when freeing the cipher handle
HMAC keys	SP 800-90A DRBG	Application memory or key database	Encrypted through key wrapping using FC_WrapKey	Automatically zeroized when freeing the cipher handle
Diffie-Hellman or EC Diffie-Hellman private components	SP 800-90A DRBG	Application memory or key database	Encrypted through key wrapping using FC_WrapKey	Automatically zeroized when freeing the cipher handle
SP 800-90A DRBG seed and entropy string	Obtained from /dev/urandom	Application memory	N/A	Automatically zeroized when seeding operation completes
SP 800-90A DRBG V and C values	Derived from the entropy string as defined in SP 800-90A	Application memory	N/A	Automatically zeroized when freeing DRBG handle
User password	Supplied by the calling application	Key database in salted form	N/A (input through API parameter)	Automatically zeroized when the module is re-initialized or overwritten when the user changes its password

Table 9: Key Management Details

6.1 Random Number Generation

The Module employs a SP 800-90A Hash_based DRBG using SHA-256 as random number generator. The Linux kernel provides /dev/urandom as a source of random numbers for DRBG seeds. Reseeding is performed by pulling more data from /dev/urandom. A product using the Module should periodically reseed the Module's random number generator with unpredictable noise by calling FC_SeedRandom. After 2^{48} calls to the random number generator the Module reseeds automatically.

The Module performs Continuous Random Number Generation Test (CRNGT) on the output of the SP800-90A DRBG to ensure that consecutive random numbers do not repeat.

In addition, the module also performs DRBG health testing as defined in section 11.3 of SP 800-90A DRBG.

6.2 Key/CSP Storage

This section identifies the cryptographic keys and CSPs that the user has access to while performing a service, and the type of access the user has.

The Module employs the following cryptographic keys and CSPs in the FIPS Approved mode of operation. Note that the private key database (key3.db/key4.db) mentioned below is within the Module's physical boundary but outside of its logical boundary.

- DSA integrity test public key: The module stores a public key for performing the power-up integrity test in the libfreebl3.chk, libnssdbm3.chk and libsoftokn3.chk files for the verification of libfreebl3.so, libnssdbm3.so and libsoftokn3.so, respectively.
- AES secret keys: The keys may be stored in memory or in the private key database (key3.db/key4.db).
- Hash_based DRBG secret values: The entropy is stored in plaintext in volatile memory. Hash_based DRBG V value (internal Hash_based DRBG state value) is stored in plaintext in volatile memory. Hash_based DRBG C value (internal Hash_based DRBG state value) is stored in plaintext in volatile memory.
- Triple-DES secret keys: The keys may be stored in memory or in the private key database (key3.db/key4.db).
- HMAC secret keys: HMAC key size must be greater than or equal to half the size of the hash function output and greater than 112 bits. The keys may be stored in memory or in the private key database (key3.db/key4.db).
- DSA/ECDSA public keys and private keys: The keys may be stored in memory or in the private key database (key3.db/key4.db).
- RSA public keys and private keys (used for digital signatures and key transport): The keys may be stored in memory or in the private key database (key3.db/key4.db).
- Diffie-Hellman/EC Diffie-Hellman public keys and private keys: The keys may be stored in memory or in the private key database.
- Authentication data (NSS User role password): Stored in salted form in the private key database (key3.db/key4.db).

Public and private keys are provided to the Module by the calling process, and are destroyed when released by the appropriate API function calls.

6.3 Key/CSP Zeroization

The Module performs explicit zeroization steps to clear the memory region previously occupied by a plaintext secret key, private key, or password. When the cipher handle is freed, the memset() function is used to zeroize memory and free() function is used to free memory allocated from the heap. A plaintext secret or private key gets zeroized when it is deleted (with a FC_DestroyObject call). All plaintext secret and private keys are zeroized when the Module is shut down (with a FC_Finalize call), or when the Module is reinitialized (with a FC_InitToken call), or when the session is closed (with a FC_CloseSession or FC_CloseAllSessions call). All zeroization is to be performed by storing the value “zeros” into every byte of the memory that is occupied by a plaintext secret key, private key or password.

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

7 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The test platform that runs the Module meets the requirements of 47 CFR FCC PART 15, Subpart B, Class A (Business use).

8 Self Tests

FIPS 140-2 requires that the Module performs self-tests to ensure the integrity of the Module and the correctness of the cryptographic functionality at start up. In addition, some functions require continuous verification, such as the random number generator. All of these tests are listed and described in this section.

8.1 Power-Up Tests

All the power-up self-tests are performed automatically without requiring any operator intervention. During the power-up self-tests no other services are available and all output is inhibited. Once the power-up self-tests are completed successfully, the Module enters operational mode and cryptographic operations are available. If any of the power-up self-tests fail, the Module enters power-up self-test error state. In error state, all output is inhibited and no cryptographic operations are allowed. The module is aborted to indicate the error. It needs to be reloaded in order to recover from the error state.

The Module implements the following Known Answer Test (KAT) and Integrity Test during the power-up:

Algorithm	Test
AES	KAT: encryption and decryption are tested separately
Triple-DES	KAT: encryption and decryption are tested separately
DSA	KAT: signature generation and signature verification are tested separately
RSA	KAT: encryption and decryption are tested separately KAT: signature generation and signature verification are tested separately
ECDSA	KAT: signature generation and signature verification are tested separately
SP800-90A Hash-based DRBG	KAT
SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512	KAT
HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384 and HMAC-SHA-512	KAT
Module integrity	DSA signature verification with 2048-bit key and SHA-256

Table 10: Module Self Tests

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

8.2 Conditional Tests

The Module implements the following Pair-wise Consistency Test (PCT) for public-private key pairs generation and Continuous Random Number Generator Test (CRNGT). If any of the conditional tests fail, the Module enters operational error state. It returns the error code CKR_DEVICE_ERROR to the calling application to indicate the error. The Module needs to be reinitialized to resume normal operation. Reinitialization is accomplished by calling FC_Finalize followed by FC_Initialize.

Algorithm	Test
DSA	PCT: signature generation and verification are tested separately
RSA	PCT: encryption and decryption are tested separately PCT: signature generation and verification are tested separately
ECDSA	PCT: signature generation and verification are tested separately
SP 800-90A DRBG	CRNGT

Table 11: Module Conditional Tests

9 Guidance

9.1 Crypto Officer Guidance

The version of the RPMs containing the FIPS validated Module is listed in section 1.3. The integrity of the RPM is automatically verified during the installation and the Crypto Officer shall not install the RPM file if the RPM tool indicates an integrity error. The RPM package of the Module can be installed by standard tools recommended for the installation of RPM packages on a SUSE Linux system (for example, rpm, yast and yast online_update).

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

Only the cipher types listed in Section 1.2 are allowed to be used.

To bring the Module into FIPS approved mode, perform the following:

1. Install the dracut-fips package:

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

2. Recreate the INITRAMFS image:

```
# dracut -f
```

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
```

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 to the kernel command line:

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

Reboot to apply these settings.

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

Because FIPS 140-2 has certain restrictions on the use of cryptography which are not always wanted, the Module needs to be put into FIPS approved mode explicitly. If the file /proc/sys/crypto/fips_enabled exists and contains a numeric value other than 0, the Module is put into FIPS approved mode at initialization time. This is the mechanism recommended for ordinary use, activated by using the fips=1 option in the boot loader, as described above.

9.1.1 Access to Audit Data

The Module may use the Unix syslog function and the audit mechanism provided by the operating system to audit events. Auditing is turned off by default. Auditing capability 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

The Module must be operated in FIPS approved mode to ensure that FIPS 140-2 validated cryptographic algorithms and security functions are used.

The following module initialization steps must be followed by the Crypto-Officer before starting to use the NSS module:

- Set the environment variable NSS_ENABLE_AUDIT to 1 before using the NSS module with an application.
- Use the application to get the function pointer list using the NSS API “FC_GetFunctionList”.
- Use the API FC_Initialize to initialize the Module. Using the FC_GetFunctionList above ensured that we selected FIPS mode, and the subsequent FC_Initialize call then initializes the module in FIPS-mode. Ensure that this returns CKR_OK. A return code other than CKR_OK means that the FIPS-mode was not enabled, and in that case, the Module must be reset and initialized again.
- For the first login, provide a NULL password and login using the function pointer C_Login, which will in-turn call FC_Login API of the Module. This is required to set the initial NSS User password.
- Now, set the initial NSS User role password using the function pointer C_InitPIN. This will call the Module's API FC_InitPIN API. Then, logout using the function pointer C_Logout, which will call the Module's API FC_Logout.
- The NSS User role can now be assumed on the Module by logging in using the User password. And the Crypto Officer role can be implicitly assumed by performing the Crypto-Officer services as listed in Section 3.1.

The Module can be configured to use different private key database formats: key3.db or key4.db. “key3.db” format is based on the Berkeley DataBase engine and should not be used by more than one process concurrently. “key4.db” format is based on SQL DataBase engine and can be used concurrently by multiple processes. Both databases are considered outside the 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 passed to the calling application only in encrypted (wrapped) form with FC_WrapKey and entered from calling application only in decrypted (unwrapped) form with FC_UnwrapKey. The cryptographic algorithms allowed for this purpose in FIPS-mode are AES, Triple-DES or RSA using the corresponding Approved modes and key sizes. Note: If the secret and private keys passed to higher-level callers are encrypted using a symmetric key algorithm, the encryption key may be derived from a password. In such a case, they should be considered to be in plaintext form in the FIPS Approved mode.

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

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

9.2.1 AES GCM Guidance

The AEC GCM IV generation is compliant with RFC 5288. The GCM block chaining mode shall only be used together with TLS protocol version 1.2 or higher. In case of power loss from the Module, the AES GCM key will be re-negotiated. No IV is stored in memory.

9.2.2 RSA and DSA Keys

The Module allows the use of 1024 bit RSA and DSA keys for legacy purposes, including signature generation.

As per SP800-131A, RSA and DSA must be used with either 2048 bit keys or 3072 bit keys. To comply with the requirements of FIPS 140-2, a user must therefore only use keys with 2048 bits or 3072 bits.

10 Mitigation of Other Attacks

The Module is designed to mitigate the following attacks.

Attack	Mitigation Mechanism	Specific Limit
Timing attacks on RSA	**RSA blinding** Timing attack on RSA was first demonstrated by Paul Kocher in 1996 [15], who contributed the mitigation code to our module. Most recently Boneh and Brumley [16] showed that RSA blinding is an effective defense against timing attacks on RSA.	None
Cache-timing attacks on the modular exponentiation operation used in RSA and DSA	**Cache invariant modular exponentiation** This is a variant of a modular exponentiation implementation that Colin Percival [17] showed to defend against cache-timing attacks.	This mechanism requires intimate knowledge of the cache line sizes of the processor. The mechanism may be ineffective when the module is running on a processor whose cache line sizes are unknown.
Arithmetic errors in RSA signatures	**Double-checking RSA signatures** Arithmetic errors in RSA signatures might leak the private key. Ferguson and Schneier [18] recommend that every RSA signature generation should verify the signature just generated.	None

11 Glossary and Abbreviations

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

12 References

[1] FIPS 140-2 Standard, http://csrc.nist.gov/groups/STM/cmvp/standards.html

[2] FIPS 140-2 Implementation Guidance, http://csrc.nist.gov/groups/STM/cmvp/standards.html

[3] FIPS 140-2 Derived Test Requirements,http://csrc.nist.gov/groups/STM/cmvp/standards.html

[4] FIPS 197 Advanced Encryption Standard, http://csrc.nist.gov/publications/PubsFIPS.html

[5] FIPS 180-4 Secure Hash Standard, http://csrc.nist.gov/publications/PubsFIPS.html

[6] FIPS 198-1 The Keyed-Hash Message Authentication Code (HMAC), http://csrc.nist.gov/publications/PubsFIPS.html

[7] FIPS 186-4 Digital Signature Standard (DSS), http://csrc.nist.gov/publications/PubsFIPS.html

[8] NIST SP 800-67 Revision 1, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, http://csrc.nist.gov/publications/PubsFIPS.html

[9] NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation: Methods and Techniques, http://csrc.nist.gov/publications/PubsFIPS.html

[10] NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, http://csrc.nist.gov/publications/PubsFIPS.html

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

[12] NIST SP 800-56A, Recommendation for Pair-Wise Key Establishment Schemes using Discrete Logarithm Cryptography (Revised), http://csrc.nist.gov/publications/PubsFIPS.html

[13] NIST SP 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, http://csrc.nist.gov/publications/PubsFIPS.html

[14] RSA Laboratories, “PKCS #11 v2.20: Cryptographic Token Interface Standard”, 2004.

http://www.cryptsoft.com/pkcs11doc/STANDARD/pkcs-11v2-20.pdf

[15] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," CRYPTO '96, Lecture Notes In Computer Science, Vol. 1109, pp. 104-113, Springer-Verlag, 1996. http://www.cryptography.com/timingattack/

[16] D. Boneh and D. Brumley, "Remote Timing Attacks are Practical," http://crypto.stanford.edu/~dabo/abstracts/ssl-timing.html

[17] C. Percival, "Cache Missing for Fun and Profit," http://www.daemonology.net/papers/htt.pdf

[18] N. Ferguson and B. Schneier, Practical Cryptography, Sec. 16.1.4 "Checking RSA Signatures", p. 286, Wiley Publishing, Inc., 2003.