Network Security Services (NSS) Cryptographic Module Version 3.12.4

FIPS 140-2 Security Policy

Level 1 Validation

Wind River Systems, Inc.

Version 1.2

Last Update: 2010-12-13

Table of Contents

List of Tables

Revision history

Version	Change date	Author(s)	Changes to previous version
1.0	2010-07-01	atsec	First version
1.1	2010-11-16	atsec	Addressed CMVP comments
1.2	2010-12-13	atsec	Addressed CMVP comments

1 Introduction

This document is the nonproprietary FIPS 140-2 security policy for the NSS Cryptographic Module to meet FIPS 140-2 level 1 requirements. This Security Policy details the secure operation of the NSS Cryptographic Library as required in Federal information Processing Standards Publication 140-2 (FIPS 140-2) as published by the National Institute of Standards and Technology (NIST) of the United States Department of Commerce.

1.1 Purpose

This security policy describes the services provided by the NSS Library cryptographic module. It provides precise specification of cryptographic security, the cryptographic module user (organization or individual operator), and the capabilities, protections, and access rights they will have when using the cryptographic module. This document also provides information about the delivery of the module and the steps required to bring it in an Approved mode of operation.

1.2 Audience

This document is required as a part of the FIPS 140-2 validation process. It describes the NSS Cryptographic Module in relation to FIPS 140-2. It is intended for security officers, developers, system administrators, and end-users.

1.3 Reference

The software module this security policy describes is a port to new platforms of the module previously certified as #1279 - The Network Security Services (NSS) Cryptographic Module (Extend ECC) Version 3.12.4, FIPS 140-2 Non- Proprietary Security Policy (validation number 1279) developed by Sun Microsystems Inc, Red Hat Inc and Mozilla Foundation Inc. The following prior NSS module specification documentation and publications are also relevant to this module:

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

[FIPS_140] Federal Information Processing Standards Publication, “FIPS PUB 140-2 Security Requirements for Cryptographic Modules”, 2002.

[FIPS_CKM] Mozila wiki, Section 7: Cryptographic Key Management

[FIPS_OE] Mozilla wiki, FIPS Operational Environment

[FIPS_FSM] Mozilla wiki, Section 4: finite Sate Model

[FIPS_MS] Mozilla wiki, FIPS Module Specification

[FIPS_PUST] Mozilla wiki, Power-Up Self-Tests

[FIPS_RS] Mozilla wiki, FIPS Roles and Services

[IG] NIST, “Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program”, 2010.

[NSS_CM3.12.4] Source code of NSS Cryptographic Module provided by Wind River Systems Inc.

[PKCS11] RSA Laboratories, “PKCS #11 v2.20: Cryptographic Token Interface Standard”, 2004. (http://www.rsasecurity.com/rsalabs/node.asp?id=2133)

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

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

[SP_1279] Sun Microsystems Inc, Red Hat Inc, Mozilla Foundation Inc, “Network Security Services (NSS) Cryptographic Module (Extend ECC) Version 3.12.4, FIPS 140-2 Non-Proprietary Security Policy Level 2 Validation”, 2010.

[SP800-57P1] NIST Special Publication 800-57, “Recommendation for Key Management – Part 1: General (Revised)”, March 2007

[TAIDHRD] 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/)

2 Cryptographic Module Specification (140-2 Section 4.1)

The following sections describe the cryptographic module specifications.

2.1 Description of Module

The NSS cryptographic module is an open-source, software only, general-purpose cryptographic library available for free under the Mozilla Public License, the GNU General Public License, and the GNU Lesser General Public License. The NSS cryptographic module consists of APIs based on the industry standard PKCS #11 version 2.20 [PKCS11]. The following table shows the overview of the security level for each of the eleven sections of validation.

Table 1 Security Level

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

2.2 Platform List

The NSS cryptographic module has been tested on the following configuration:

- Wind River Linux Secure 1.0

The module has been tested on the following platforms:

Table 2 Platforms

Model	Operating System and Version
x86_64 Nehalem Xeon 5500	Wind River Linux Secure 1.0
x86_64 Pentium core2 duo	Wind River Linux Secure 1.0

2.3 Modes of Operation

The NSS cryptographic module operates in two modes of operations: a FIPS approved mode and a non-FIPS approved mode. If the NSS cryptographic module is initialized by calling the standard PKCS #11 function C_GetFunctionList and the API functions are called via the function pointers in that list, then the application selects the non-FIPS approved mode by default. In non-FIPS mode, the non-approved API function NSC_Initilize initializes the PKCS #11 library and NSC_Finalize indicates that an application is done with the PKCS #11 library.

In order to operate in the FIPS approved mode, the application must follow the security rules listed in Section 11.2.1 and initialize the module properly. The application must call the API functions via an alternative set of function pointers - see Section 11.2.1 for details.

2.4 Description of FIPS Approved Mode

In the FIPS approved mode, the module will support the following FIPS approved security function:

Table 3 FIPS approved cryptographic algorithms with certificate numbers

Cryptographic Algorithm	**Validation Certificate#**	**Usage**	**Keys/CSPs**
Triple-DES (SP 800-67, Appendix E OF SP 800-38A)	949	TECB(e/d; KO 1,2); TCBC(e/d; KO 1,2)	Secret Key
AES (FIPS 197, SP 800-38A)	1374	ECB(e/d; 128,192,256); CBC(e/d; 128,192,256)	Secret Key

DSA
(FIPS 186-2 with Change Notice 1)
	450	PQG(gen) MOD(1024);
		PQG(ver) MOD(1024);
		KEYGEN(Y) MOD(1024);
		SIG(gen) MOD(1024);
		SIG(ver) MOD(1024);
			Private and public key

RSA (PKCS#1 v2.1)	673	ALG[RSASSA-PKCS1_V1_5]; SIG(gen); SIG(ver); 1024 , 1536 , 2048 , 3072 , 4096 , SHS: SHA-1 , SHA-256 , SHA-384 , SHA-512	Private and public key

ECDSA
(FIPS 186-2 with Change Notice 1)
	174	PKG: CURVES(P-256 P-384 P-521)
		PKV: CURVES(P-256 P-384 P-521)
		SIG(gen): CURVES(P-256 P-384 P-521)
		SIG(ver): CURVES(P-256 P-384 P-521)
			Private and public key

SHS (FIPS 180-3)	1256	SHA-1 (BYTE-only) SHA-256 (BYTE-only) SHA-384 (BYTE-only) SHA-512 (BYTE-only)	N/A
HMAC (FIPS 198)	807	HMAC-SHA1 (Key Sizes Ranges Tested: KS<BS KS=BS KS>BS) HMAC-SHA256 (Key Size Ranges Tested: KS<BS KS=BS KS>BS) HMAC-SHA348 (Key Size Ranges Tested: KS<BS KS=BS	Secret key

		KS>BS) HMAC-SHA512 (Key Size Ranges Tested: KS<BS KS=BS KS>BS)	
DRBG (NIST SP 800-90)	49	[Hash_DRBG: SHA-256]	Seed

The module will support the following non-approved cryptographic functions:

Table 4 Non-approved cryptographic functions

Algorithm	**Validation Certificate**	**Usage**	**Keys/CSPs**
MD5	N/A	Hashing	N/A
MD2	N/A	Hashing	N/A
RC2	N/A	Encryption and Decryption	Symmetric key
RSA	N/A	Encryption and Decryption for key wrapping	Asymmetric keys
RC4	N/A	Encryption and Decryption	Symmetric key
DES	N/A	Encryption and Decryption	Symmetric key
SEED	N/A	Encryption and Decryption	Symmetric key
CAMELLIA	N/A	Encryption and Decryption	Symmetric key
ECDSA	N/A	ANSI X9.62-1998 binary, SEC 2 prime, SEC 2 binary curves - see Section 2.4.1 for a complete list of the non-NIST approved curves	Asymmetric keys

Warning: Users should not use these non-approved cryptographic functions in the FIPS approved mode of operation.

The NSS cryptographic module uses the following key establishment techniques in the FIPS approved mode of operation:

Table 5 Asymmetric Establishment techniques allowed in a FIPS approved mode

Algorithm	**Key size**	**Strength of mechanism/ algorithm [bits of security]**
Diffie-Hellman (key agreement)	public key of sizes 1024-2236 bits	see caveat below
ECDH (key agreement)	elliptic curve key sizes of 163-571 bits	see caveat below
RSA (PKCS #1, key wrapping)	RSA key sizes of 1024-8192 bits	see caveat below

Note: Refer to Section 5.6.1 of [SP800-57P1] for information regarding the strengths of security of the asymmetric key establishment techniques.

CAVEATS:

Since the NSS cryptographic module allows a key establishment method to establish a cryptographic key that is stronger than the key establishment method, the following warnings are required by Section 7.5 of the implementation guidelines of [IG]:

1. Diffie-Hellman (key agreement, key establishment methodology provides between 80 bits and 112 bits of encryption strength)
2. ECDH (key agreement, key establishment methodology provides between 80 bits and 256 bits of encryption strength)
3. RSA (PKCS #1, key wrapping, key establishment methodology provides between 80 bits and 192 bits of encryption strength)
4. MD5 for use in TLS only.

2.4.1 Non-NIST Recommended Elliptic Curve

The Extend ECC version of NSS cryptographic module implements the NIST recommended as well as non-NIST recommended curves.

Table 6 Non-NIST Recommended Curves

Curve Family	Curve Name
ANSI X9.62-1998 binary curves	c2pnb163v1, c2pnb163v2, c2pnb163v3, c2tnb191v1, c2tnb191v2, c2tnb191v3, c2tnb239v1, c2tnb239v2, c2tnb239v3,c2tnb359v1, and c2tnb431r1
SEC 2 prime curves	secp160k1, secp160r1, secp160r2, secp192k1, secp224k1, and secp256k1
SEC 2 binary curves	sect163r1, sect193r1, sect193r2, and sect239k1

Although Section 1.6 of FIPS 140-2 Implementation Guidance permits the use of non-NIST recommended curves in the FIPS-Approved mode, we recommend that these curves should not be used in the FIPS approved mode of operation.

The following table lists non-NIST recommended curves implemented by the Extend ECC version of the NSS cryptographic module, which **are not to** be used in the FIPS approved mode.

2.5 Cryptographic Module Boundary

The physical module boundary is a surface of the case of the platform. The logical module boundary is depicted in the software block diagram.

2.5.1 Physical Cryptographic Boundary

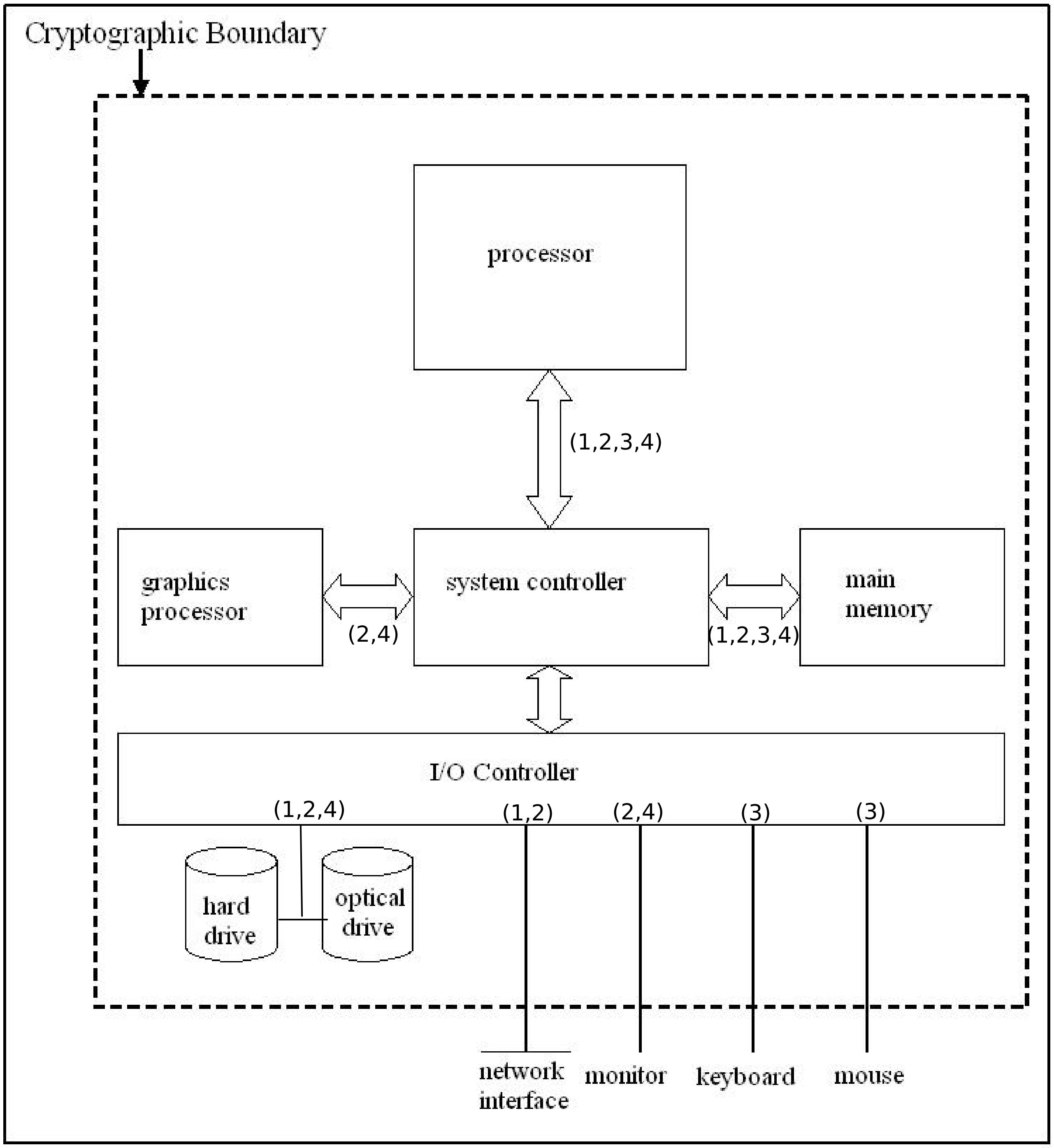

Figure 1 Physical Cryptographic boundary diagram

For figure 1, the labels 1, 2, 3, and 4 correspond to Data In, Data Out, Control In, and Status Out dataflows, respectively.

2.5.2 Logical Cryptographic Boundary

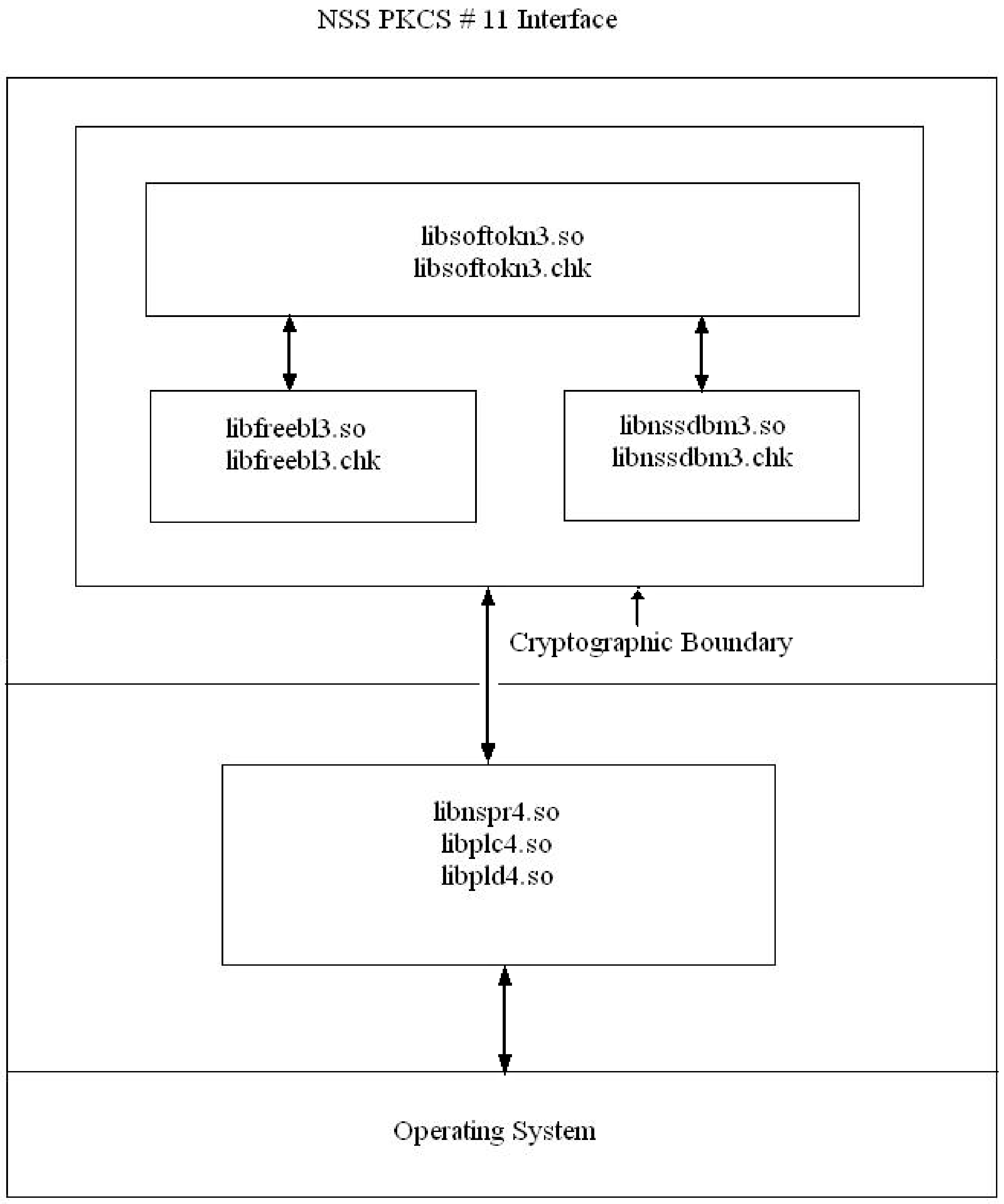

Figure 2 Logical Cryptographic boundary diagram

The arrows in Figure 2 depict the flow of information throughout the module. Since the interfaces between the module and the external operating environment, and the interfaces between the internal components of the module are all programmatic interfaces, the control, data, and status inputs and outputs flow closely in parallel. This is the case since each function implemented in the module has input and output parameters that pass control, data, and status information through the API interfaces.

The NSS cryptographic module requires the Netscape Portable Runtime (NSPR) libraries. NSPR provides a cross-platform API for non-GUI operating system facilities, such as threads, thread synchronization, normal file and network I/O, interval timing and calendar time, atomic operations, and shared library linking. NSPR also provides utility functions for strings, hash tables, and memory pools.

NSPR consists of the following shared libraries/DLLs:

- libplc4.so
- libplds4.so
- libnspr4.so

NSPR is outside the cryptographic boundary because none of the NSPR functions are security-relevant.

3 Cryptographic Module Ports and Interfaces (140-2 Section 4.2)

The NSS Cryptographic module is a software implementation only. This means it does not include any hardware or firmware components. All the input to the module is processed via function arguments. All the output is returned to the caller either as return codes or as updated memory objects pointed to by some of the arguments.

3.1 Physical Interface

The physical interfaces of the NSS cryptographic module are the physical ports, physical covers, doors or openings, manual controls, and physical status indicators of the general purpose computer it runs on.

3.2 Logical Interfaces

The logical interfaces use library functions which exchange the encrypted data, keys, and all control information through calls. The module uses different input and output function arguments to differentiate between data input, data output, control input and status output. Similarly, it uses different buffers for input and output data. The module performs zeroization of the input buffer that contains security related information.

The NSS cryptographic module includes the following logical interfaces:

- Data input interface: Password, cryptographic keys (encrypted or plaintext), initialization vectors, plaintext data, cipher text, and signed data supplied to and processed by a cryptographic module is via function input argument.
- Data output interface: Cryptographic keys (encrypted or plaintext), initialization vectors, plaintext data, cipher text, and digital signatures received via function output argument from the module.
- Control input interface: The control interfaces consist of control data such as algorithms, module settings, commands specified by input arguments, and function calls to control the operation of the module.
- Status output interface: This interface indicates the status of the module such as function return codes, error codes, or output arguments.

3.2.1 PKCS #11 (Cryptoki) API

PKCS# 11 (Cryptoki) API is one of the logical interfaces of the NSS cryptographic module that provides access to the module. The module has three PKCS # 11 tokens. The FIPS PKCS # 11 token allows applications using the NSS cryptographic module to operate in a strictly FIPS mode. Functions of FIPS PKCS# 11 API are listed in Table 8. The other two tokens implement the non-FIPS approved mode of operation.

Data Output in Error State

All data via the data output interface is restricted when the NSS cryptographic module is in the error state. The error state is tracked by the Boolean state variable `sftk_fatalError`. All the PKCS # 11 function that output data via the data output interface use this state variable and, if it is true, return the `CKR_DEVICE_ERROR` error code.

In the error state, the functions which are responsible for restart, shutdown, and re-initialization of the module or output status information can be invoked. These functions are as follows:

- `FC_GetFunctionList`
- `FC_Initilize`
- `FC_Finalize`
- `FC_GetSlotList`

- `FC_GetSlotInfo`
- `FC_GetTokenInfo`
- `FC_InitToken`
- `FC_CloseSession`
- `FC_CloseAllSessions`
- `FC_WaitForSlotEvent`
- `FC_GetInfo`

Descriptions of above listed functions are provided in Table 8.

3.2.2 Data Output during Self test

All data via the data output interface is suppressed when `FC_Initialize` function of the NSS cryptographic module is performing self-tests.

4 Roles, Services, and Authentication (140-2 Section 4.3)

The following sections describe the authorized roles, services associated with the NSS module and the authentication policy.

4.1 Roles

The NSS cryptographic module supports the two authorized roles for operators i.e. NSS user and a Crypto Officer. This security policy introduces a new implicit role called "Everyone" and it is responsible for all the public services specified in Section 4.3.1. Please note that this role is not defined in [FIPS-RS].

Table 7 Roles

Role	Responsibilities and Services (see list provided in Section 4.2)
NSS User	Utilizes secure services Provides access to all cryptographic and general- purpose services (except installation of module) Provides access to all keys stored in the private key database Retrieval, updating, and deletion of keys from the private key database
Crypto Officer	Installation of module Control the access before and after the installation Management of physical access to the computer, execution of NSS cryptographic module code Management of security facilities provided by the operating system
Everyone (not an authorized role for operators)	Responsible for Public Services such as module initialization Random number generation Parallel function management

The NSS cryptographic module uses a role-based approach for accessing services – by authenticating to the module, an operator assumes a role and gains access to the services accessible by that role. Please refer to section 4.4 for information on role-based operator authentication.

4.2 Specification of Maintenance Role

The NSS cryptographic module does not have a maintenance role. Hence, this section is not applicable.

4.3 Services

The following section describes the approved services with respect to the applicable FIPS 140-2 requirements.

4.3.1 Approved Services

The NSS Cryptographic module consists of the following types of services:

- Public Services: The public services do not require user authentication and/or access to CSPs. Message digesting services are public only when CSPs are not accessed. These services are mapped to the role "Everyone".
- Private Services: The private services require user authentication as these services access CSPs (for example, `FC_GenerateKey`, `FC_GenerateKeyPair`)

See Table 8 for specification of the services. It contains each service as an API function, associated role, service type, and type of access to the cryptographic keys and Critical Security Parameters (CSPs). CSPs contain security related information (for example secret and private cryptographic keys, and authentication data such as passwords and PINs) whose disclosure or modification can compromise the security of NSS cryptographic module.

The access types are denoted as follows:

- **'R'** stands for Read
- **'W'** stands for Write
- **'Z'** stands for Zeroize

Table 8 Approved Services

Service Category	Role	Function Name	Description	CSPs	Access type
FIPS 140-2 specific	Everyone	`FC_GetFunctionList`	In FIPS approved mode, it returns the list of function pointers	none	-
Module Initialization	Everyone	`FC_InitToken`	Initializes or reinitializes a token	password and all keys	Z
	Crypto Officer	`FC_InitPIN`	Initializes the user's password (sets the user's initial password)	password	W
General Purpose	Everyone	`FC_Initialize`	Initializes the module library for the FIPS Approved mode of operation. This function provides the power-up self-test service.	none	-
	Everyone	`FC_Finalize`	Finalizes (shuts down) the module library	all keys	Z
	Everyone	`FC_GetInfo`	Obtains general information about the module library	none	-
Slot and Token Management	Everyone	`FC_GetSlotList`	Obtains a list of slots in the system	none	-
	Everyone	`FC_GetSlotInfo`	Obtains information about a particular slot	none	-
	Everyone	`FC_GetTokenInfo`	Obtains information about the token. This function provides the Show Status service.	none	-
	Everyone	`FC_WaitForSlotEvent`	This function is not supported by the NSS cryptographic module.	none	-

Service Category	Role	Function Name	Description	CSPs	Access type
	Everyone	`FC_GetMechanismList`	Obtains a list of mechanisms (cryptographic algorithms) supported by a token	none	-
	Everyone	`FC_GetMechanismInfo`	Obtains information about a particular mechanism	none	-
	NSS User	`FC_SetPIN`	Changes the user's password	password	RW
Session Management	Everyone	`FC_OpenSession`	Opens a connection (session) between an application and a particular token	none	-
		`FC_CloseSession`	Closes a session	keys of the session	Z
	Everyone	`FC_CloseAllSessions`	closes all sessions with a token	all keys	Z
	Everyone	`FC_GetSessionInfo`	Obtains information about the session. This function provides the Show Status service.	none	-
	Everyone	`FC_GetOperationState`	Saves the state of the cryptographic operation in a session. This function is only implemented for message digest operations.	none	-
	Everyone	`FC_SetOperationState`	Restores the state of the cryptographic operation in a session. This function is only implemented for message digest operations.	none	-
	Everyone	`FC_Login`	Logs into a token	password	R
	NSS User	`FC_Logout`	Logs out from a token	none	-
Object Management	NSS User	`FC_CreateObject`	Creates an object	original key	R
	NSS User	`FC_CopyObject`	Creates a copy of an object	new key	W
	NSS User	`FC_DestroyObject`	Destroys an object	key	Z

Service Category	Role	Function Name	Description	CSPs	Access type
	NSS User	`FC_GetObjectSize`	Obtains the size of an object in bytes	key	R
	NSS User	`FC_GetAttributeValue`	Obtains an attribute value of an object	key	R
	NSS User	`FC_SetAttributeValue`	Modifies an attribute value of an object	key	W
	NSS User	`FC_FindObjectsInit`	Initializes an object search operation	none	-
	NSS User	`FC_FindObjects`	Continues an object search operation	keys matching the search criteria	R
	NSS User	`FC_FindObjectsFinal`	Finishes an object search operation	none	-
Encryption and Decryption	NSS User	`FC_EncryptInit`	Initializes an encryption operation	encryption key	R
	NSS User	`FC_Encrypt`	Encrypts single-part data	encryption key	R
	NSS User	`FC_EncryptUpdate`	Continues a multiple part encryption operation	encryption key	R
	NSS User	`FC_EncryptFinal`	Finishes a multiple part encryption operation	encryption key	R
	NSS User	`FC_DecryptInit`	Initializes a decryption operation	encryption key	R
	NSS User	`FC_Decrypt`	Decrypts single-part encrypted data	encryption key	R
	NSS User	`FC_DecryptUpdate`	Continues a multiple part decryption operation	encryption key	R
	NSS User	`FC_DecryptFinal`	Finishes a multiple part decryption operation	encryption key	R
Message Digesting	Everyone	`FC_DigestInit`	Initializes a message digesting operation	none	-
	Everyone	`FC_Digest`	Digests single-part data	none	-
	Everyone	`FC_DigestUpdate`	Continues a multiple part digesting operation	none	-
	NSS	`FC_DigestKey`	Continues a multipart message	key	R

Service Category	Role	Function Name	Description	CSPs	Access type
	User (see the note at the end of the table)		digesting operation by digesting the value of a secret key as part of the data already digested		
	Everyone	`FC_DigestFinal`	Finishes a multiple part digesting operation	none	-
Signature and Verification	NSS User	`FC_SignInit`	Initializes a signature operation	signing/HMAC key	R
	NSS User	`FC_Sign`	Signs single-part data	signing/HMAC key	R
	NSS User	`FC_SignUpdate`	Continues a multiple part signature operation	signing/HMAC key	R
	NSS User	`FC_SignFinal`	Finishes a multiple part signature operation	signing/HMAC key	R
	NSS User	`FC_SignRecoverInit`	Initializes a signature operation, where the data can be recovered from the signature	RSA signing key	R
	NSS User	`FC_SignRecover`	Signs single-part data, where the data can be recovered from the signature	RSA signing key	R
	NSS User	`FC_VerifyInit`	Initializes a verification operation	Verification/HMAC key	R
	NSS User	`FC_Verify`	Verifies a signature on single-part data	verification/HMAC key	R
	NSS User	`FC_VerifyUpdate`	Continues a multiple part verification operation	verification/HMAC key	R
	NSS User	`FC_VerifyFinal`	Finishes a multiple part verification operation	RSA verification key	R
	NSS User	`FC_VerifyRecover`	Verifies a signature on single-part data, where the data is recovered from the signature	RSA verification key	R
Key Management	NSS User	`FC_GenerateKey`	Generates a secret key (used by TLS to generate premaster	key	W

Service Category	Role	Function Name	Description	CSPs	Access type
			secrets)		
	NSS User	`FC_GenerateKeyPair`	Generates a public/private key pair. This function performs the pair wise consistency tests.	key pair	W
	NSS User	`FC_WrapKey`	Wraps (encrypts) a key	wrapping key	R
				key to be wrapped	R
	NSS User	`FC_UnwrapKey`	Unwraps (decrypts) a key	Unwrapping key	R
				Unwrapping key	W
	NSS User	`FC_DeriveKey`	Derives a key from a base key (used by TLS to derive keys from the master secret)	base key	R
				derived key	W
Dual-function cryptographic operations	NSS User	`FC_DigestEncryptUpdate`	Continues a multiple part digesting and encryption operation	encryption key	R
	NSS User	`FC_DecryptDigestUpdate`	Continues a multiple part decryption and digesting operation	decryption key	R
	NSS User	`FC_SignEncryptUpdate`	Continues a multiple part signing and encryption operation	signing/HAMC key	R
				encryption key	R
	NSS User	`FC_DecryptVerifyUpdate`	Continues a multiple part decryption and verify operation	decryption key	R
				verification/ HMAC key	R
Random Number Generation	Everyone	`FC_SeedRandom`	Mixes in additional seed material to the random number generator	mixed-in seed	RW
	Everyone	`FC_GenerateRandom`	Generates random data. This function performs the continuous random number generator test.	initial seed, random values for key material	RW
Parallel Function Management	NSS User	`FC_GetFunctionStatus`	A legacy function, which simply returns the value 0x00000051 (function not	none	-

Service Category	Role	Function Name	Description	CSPs	Access type
			parallel)		
	NSS User	`FC_CancelFunction`	A legacy function, which simply returns the value 0x00000051 (function not parallel)	none	-

Note: The message digesting functions (except `FC_DigestKey`) don't require the user to assume an authorized role because they do not use any keys. `FC_DigestKey` computes the message digest (hash) of the value of a secret key; therefore the user needs to assume the NSS User role for this service.

4.4 Operator Authentication

The following sections describe the authentication policy of the NSS Cryptographic module.

4.4.1 Role-Based Authentication

The NSS cryptographic module uses role-based authentication to control access to the NSS module. To perform sensitive services using NSS cryptographic module, the user must log into the module and perform authentication procedure using a password. This password is used to encrypt and decrypt private key of the user. However, a discretionary access control is used to protect all other information (for example, the public key certificate database). See Section 8.2.2 for details regarding the discretionary access control.

Note: For Security Level 1, a cryptographic module is not required to employ authentication mechanisms to control access to the module.

4.4.2 Clearing of Previous Authentications on Power Off

When the process accessing the NSS cryptographic module terminates or the general purpose computer is powered off, the result of authentications in the memory are automatically cleared.

4.4.3 Protection of Authentication Data

The NSS cryptographic module stores a verifier for the user’s password in the NSS key database. The module verifies the password by deriving a Triple-DES key from the password, using an extension of the PKCS #5 PBKDF1 key derivation function with an 16-octet salt, an iteration count of 1, and SHA-1 as the underlying hash function, decrypting the stored encrypted password check-string with the Triple-DES key, and comparing the decrypted string with the known password check-string. It is computationally infeasible to acquire a password from the verifier. This mechanism protects against unauthorized disclosure and modification of the user’s password.

4.4.4 Initialization of Authentication Mechanism

The operator implicitly assumes a Crypto Officer role when installing the NSS cryptographic module library files. Once the library files are installed, the Crypto Officer calls the function FC_InitPIN to set the operator's initial password. Please note that it is not necessary to call FC_InitToken to initialize the NSS cryptographic module. The NSS cryptographic module is initialized automatically when FC_Initialize is called for the first time. The Crypto Officer may call FC_InitToken to re-initialize the NSS cryptographic module.

When the NSS cryptographic module is accessed for the first time, it does not use a factory-set or default password to authenticate the operator. Hence, login mechanism of the general purpose computer is used to control access to the module before it is initialized. If the general purpose computer is not protected with a system login password, procedural controls or physical access control must be used to control access to the computer before the module is initialized.

4.4.5 Change of Authentication Data

A Function FC_SetPIN is called with both the old password and new password as arguments to change the password by the NSS user.

4.4.6 Strength of Authentication Mechanism

The NSS cryptographic module enforces the following requirements on password change or initialization in the FIPS approved mode:

- The password must be **seven** characters long
- The password must contain characters from three or more character classes. There are five characters classes: digits (0-9), ASCII lowercase letters, ASCII uppercase letters, ASCII non-alphanumeric characters (such as space and punctuation marks), and non-ASCII characters.

  Note: If the first character of the password is an ASCII uppercase, or the last character of the password is a digit, then neither the uppercase letter nor the digit is counted towards its character class.

Probability of guessing the correct password randomly

Assumptions:

1. The password is at least seven characters long and the characters of the password are independent with each other.
2. The probability of guessing an individual character of the password is less than 1/10.

The probability of a successful password guess is less than (1/10) ^ 7= 1/10,000,000.

In the FIPS approved mode, after each failed authentication attempt, the NSS cryptographic module introduces a one-second delay before returning to the caller. This mechanism allows at most 60 authentication attempts during a one minute period. Therefore, the probability of a successful password guess is less than 0.6* (1/100,000).

4.4.7 Concurrent Operators

The NSS cryptographic module does not allow concurrent operators. Please note that on a multi-user operating system; this rule is enforced by making the NSS certificate and private key databases readable and writable to the owner of the files only.

5 Access Control Policy

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

5.1 Cryptographic Keys and CSPs

In the FIPS- Approved mode, the NSS cryptographic module employs the following cryptographic keys and CSPs:

- AES secret keys (128-bit, 192-bit and 256-bit) may be stored in the memory or private database (key3.db, key4.db)
- Triple-DES secret keys (168-bit) may be stored in the memory or private database (key3.db, key4.db)
- DSA public and private keys (1024-bit) may be stored in the memory or private database (key3.db, key4.db)
- Diffie-Hellman public and private keys (1024-2236 bit) may be stored in the memory or private database (key3.db, key4.db)
- RSA public and private keys (1024-2236 bit) may be stored in the memory or private database (key3.db, key4.db)
- EC public and private keys (160 bits or higher) may be stored in the memory or private database (key3.db, key4.db)
- HMAC keys (size must be greater than or equal to half the size of hash function output) may be stored in the memory or private database (key3.db, key4.db)
- Hash_DRBG (SHA 256): Hash DRBG entropy - 880-bit value externally-obtained for module DRBG; stored in plaintext in volatile memory. Hash DRBG V value - Internal Hash DRBG state value; stored in plaintext in volatile memory. Hash DRBG C value - Internal Hash DRBG state value; stored in plaintext in volatile memory
- TLS pre-master secret used in deriving the TLS master secret (48 byte) and TLS master secret shared between the peers in TLS connections, used in the generation of symmetric cipher keys, IVs, and MAC secrets for TLS (48 byte) may be stored in the memory
- Authentication data (passwords) may be stored in the private key database (key3.db, key4.db)

The module stores all cryptographic keys, CSPs, and plaintext data in the NSS database. The module permits only the owner of the files to read and modify the NSS database with proper authentication.

Note: The private key database (key3.db, key4.db) mentioned above are outside the NSS cryptographic boundary. Also, the NSS module does not implement the TLS protocol. Rather, it implements the cryptographic mechanism, such as the TLS-specific key generation and derivation operations, which can be used to implement the TLS protocol.

6 Finite State Model

The state transition diagram of NSS cryptographic module is shown below:

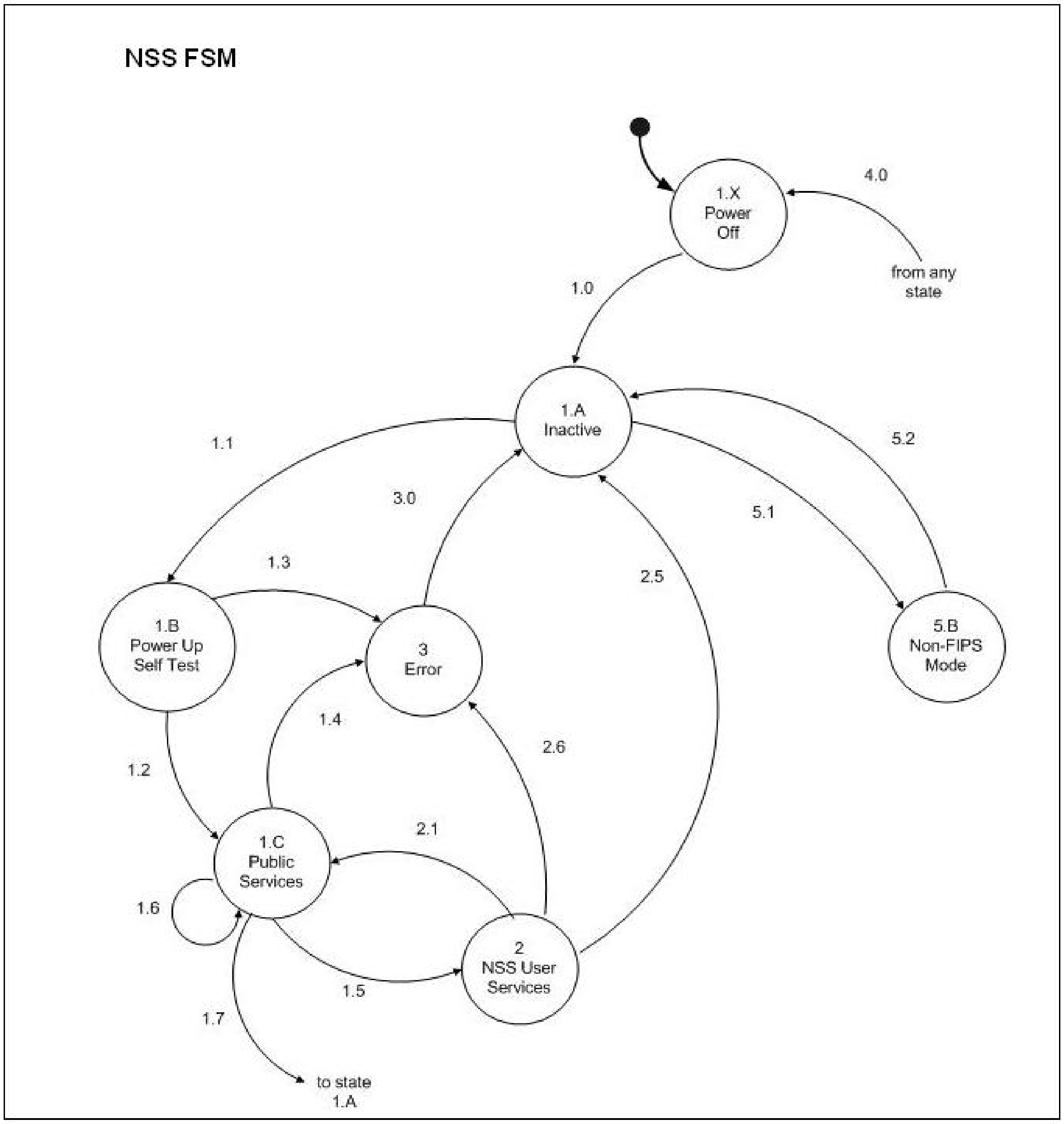

Figure 6.1 State transition diagram of NSS cryptographic module

In the FIPS-Approved mode, when the application calls the `FC_Initialize` function of the NSS cryptographic module, the state changes and power up self tests are performed. If the self-test is successful, the module is considered to be initialized and enters an operational state of the FIPS approved mode. Please refer to Section 10 for a description of power up self test. If the module enters an error state, then in order to recover from the error state, it needs to shutdown (transition 3.0) and re-initialized (transition 1.1).

If the data and control inputs are valid and the module performs the service successfully, the module outputs the requested data or status information and returns `CKR_OK`. If the data and control inputs are invalid or the module encounters an error (for example, running out of memory) when performing a service, the module does not output any data and simply returns an appropriate error code (for example, `CKR_HOST_MEMORY`, `CKR_TOKEN_WRITE_PROTECTED`, `CKR_TEMPLATE_INCOMPLETE`, or `CKR_ATTRIBUTE_VALUE_INVALID`).

The following table shows various states of NSS cryptographic module:

Table 9 States

State Label	State Mnemonic	State Descriptor	Distinct Indicator
1.X	Power Off	Host computer is powered off. The initial state	Host computer's power light is off.
1.A	Inactive	Host computer is up and running.	Host computer's power light is on.
1.B	Power up Self test	NSS cryptographic module library initialization for the FIPS approved mode has been initiated. This state performs library initialization, software integrity test, and power-up self-tests.	The `FC_Initialize` call is executing.
1.C	Public Services	NSS cryptographic module library has been initialized for the FIPS approved mode and its self-tests have passed. Services that do not require logging in to the module are available.	Public services can be invoked. Private services fail with the error code `CKR_USER_NOT_LOGGED_IN`.
2	NSS User Services	Operator has successfully logged in to assume the NSS User role and has access to all the services provided by the FIPS approved mode of the NSS cryptographic module.	All services can be invoked.
3	Error	The FIPS approved mode of the NSS cryptographic module either has failed a conditional test while performing a service or has failed a power-up or operator-initiated self-test. No further cryptographic operations will be performed.	Only `FC_Finalize`, `FC_InitToken`, `FC_CloseSession`, `FC_CloseAllSessions`, `FC_WaitForSlotEvent`, and the "get info" functions (`FC_GetFunctionList`, `FC_GetInfo`, `FC_GetSlotList`, `FC_GetSlotInfo`, and `FC_GetTokenInfo`) can be invoked. `FC_Initialize` fails with the error code `CKR_CRYPTOKI_ALREADY_INITIALIZED`. All other functions fail with the error code `CKR_DEVICE_ERROR`.
5.B	Non- FIPS mode	The non-FIPS approved mode of the NSS cryptographic module has been activated. This is a composite state whose sub-states are not relevant to FIPS 140-2.	`NSC_Initialize` has been called successfully. All other `NSC_xxx` functions may be called.

Table 10 Transitions

Transition#	Current State	Next State	Input Event	Output Event
1.0	Power Off	Inactive	Host computer is powered up	None
1.1	Inactive	Power Up Self Test	`FC_Initialize` called	Opens the databases. Power-up self-tests initiated.
1.2	Power Up Self Test	Public Services	Successful library initialization, software integrity test, and power-up self-tests	`FC_Initialize` sets the internal Boolean state variable `sftk_fatalError` to false and returns `CKR_OK`
1.3	Power Up Self Test	Error	Software integrity test or power-up self-test failure	`FC_Initialize` sets the internal Boolean state variable `sftk_fatalError` to true and returns `CKR_DEVICE_ERROR`
1.4	Public Services	Error	Conditional self-test (continuous random number generator test) failed while performing a service (random number generation)	The function (`FC_SeedRandom` or `FC_GenerateRandom`) sets the internal Boolean state variable `sftk_fatalError` to true and returns `CKR_DEVICE_ERROR`
1.5	Public Services	NSS User Services	User login succeeded	`FC_Login` sets the internal Boolean state variable `isLoggedIn` to true and returns `CKR_OK`
1.6	Public Services	Public Services	User login failed	`FC_Login` returns a nonzero error code (for example, `CKR_PIN_INCORRECT`)
1.7	Public Services	Inactive	`FC_Finalize` called	`FC_Finalize` returns `CKR_OK`
2.1	NSS User Service	Public Services	User logout requested	`FC_Logout` sets the internal Boolean state variable `isLoggedIn` to false and returns `CKR_OK`
2.5	NSS User Service	Inactive	`FC_Finalize` called	`FC_Finalize` returns `CKR_OK`
2.6	NSS User Service	Error	Conditional self-test (continuous random number generator test or pair-wise consistency test) failed while performing a service (random number generation or key pair generation)	The function (`FC_SeedRandom`, `FC_GenerateRandom`, or `FC_GenerateKeyPair`) sets the internal Boolean state variable `sftk_fatalError` to true and returns `CKR_DEVICE_ERROR` or `CKR_GENERAL_ERROR`
3.0	Error	Inactive	`FC_Finalize` called	`FC_Finalize` returns `CKR_OK`
4.0	Any State other than “Power Off”	Power Off	Host computer is powered off	None

Transition#	Current State	Next State	Input Event	Output Event
5.1	Inactive	Non-FIPS mode	`NSC_Initialize` called	Opens the databases. `NSC_Initialize` returns `CKR_OK`
5.2	Non-FIPS mode	Inactive	`NSC_Finalize` called	`NSC_Finalize` returns `CKR_OK`

7 Physical Security (140-2 Section 4.5)

The NSS cryptographic module is a security level 1 software module and offers no physical security.

8 Operational Environment (140-2 Section 4.6)

The following sections describe the operational environment of the NSS cryptographic module.

8.1 Applicability

The NSS cryptographic module has a general purpose, modifiable operational environment. For security level 1, it uses the following commercially available operating systems:

Wind River Linux Secure 1.0

8.2 Solution

8.2.1 Single Operator mode of Operation

In order to use the NSS cryptographic module at security level 1, there may be multiple users, but only one user at a time should use the module. For a note on concurrent users, please refer to Section 4.4.7.

8.2.2 Configuration of Discretionary Access Control

In order to restrict access to stored cryptographic software and programs, the user should set the file mode permissions so that all users can execute the library files, but only the files' owner can modify the files (write, replace, and delete) during the installation of the NSS cryptographic Library. For more information regarding access to the cryptographic keys, CSPs, and plaintext data, refer to Section 5.1.

8.2.3 Software Integrity Test

Refer to Section 10.1.2.

9 Cryptographic Key Management (140-2 Section 4.7)

The following sections describe the key management of the NSS cryptographic module.

9.1 Key/CSP Storage

The NSS cryptographic module does not store any password (for example, the password for password-based encryption, or the private key database password) on the disk in plaintext. At security level 1, the module limits the operating system to a single operator mode of operation. The cryptographic keys are stored as follows:

Table 11 Key Storage

Type of key	Storage
Private and secret keys	Private key database
Public keys and certificate	Private key and certificate database
Temporary (session) keys	Memory (RAM)

The OS protects all the cryptographic keys stored in the private key database and certificate database from unauthorized disclosure, modification, and substitution. When the public keys are stored in the memory, the OS protects them from unauthorized disclosure, modification, and substitution.

The following keys are used internally by the module and are not visible to the operator:

- The Triple-DES key used to encrypt the secret keys and private keys is derived from the user's password and it is stored in the private key database.
- The 1024-bit DSA public keys for the software integrity test are stored along with the DSA signatures in the `.chk` files for the softoken (PKCS #11), libnssdbm3, and freebl shared libraries/DLLs. The DSA domain parameters (prime p, subprime q, base g) and public key (y) are stored in a straight binary format (not DER encoded).

9.2 Key and CSP List

Refer to Section 5.1 for the list of cryptographic keys and CSPs.

9.3 Key/CSP Generation

The NSS cryptographic module uses the `FC_GenerateKey` function to generate secret keys and domain parameters, and the `FC_GenerateKeyPair` function to generate public/private key pairs. The NSS cryptographic module generates keys with at most 256 bits of security (refer to table 2 in Section 5.6.1 of NIST Special Publication (SP) 800-57 Part 1), Therefore, compromising the security of the key generation method (for example, guessing the seed value to initialize the Approved RNG) requires at least as many operations as determining the value of the generated key.

The following approved key generation methods are used by the module:

- The Approved RNG specified as Algorithm Hash_DRBG of SP 800-90 is used to generate cryptographic keys (for example, secret keys for symmetric key algorithms and HMAC) used by the approved and non-approved security functions.
- DSA public and private keys are generated using the method specified in FIPS 186-2 with Change Notice 1.
- RSA public and private keys are generated using the method specified in PKCS #1.
- ECDSA public and private keys are generated using the method specified in ANSI X9.62-1998.
- The prime numbers that are generated for both RSA and DSA are tested using the Miller-Rabin test (FIPS 186-2 Appendix 2.1. A Probabilistic Primality Test).

9.4 Key/CSP Establishment

Please refer to Table 5 in Section 2.4 for information regarding Key/CSPs establishment techniques.

9.5 Key/CSP Entry and Output

The NSS cryptographic module does not use either manual or electronic key entry and output methods or support entry of the seed key during key generation. The module uses the following automated key transport methods:

- The `FC_UnwrapKey` function enters an encrypted secret or private key into the module.
- The `FC_WrapKey` function outputs an encrypted secret or private key from the module.

In the FIPS approved mode of operation, the encrypted secret and private keys, entered into or output from the module are encrypted using one of the following approved algorithms:

- Triple-DES
- AES
- Key Wrapping using RSA keys

Note: A password based encryption is not FIPS approved. Hence, the AES or Triple-DES key derived from a password, the encrypted secret or private key is considered to be in plaintext form.

9.6 Key/CSP Zeroization

The NSS cryptographic module performs key zeroization to clear the memory area pre-occupied by the private key, secret key and password. The passwords are automatically zeroized by the module after use. All plaintext secret and private keys are zeroized when:

- The module is shutdown with a `FC_Finalize` call
- The module switches between the FIPS and non-FIPS modes with a `NSC_Finalize`/`FC_Initialize` or `FC_Finalize`/`NSC_Initialize` call sequence
- The module is reinitialized with a `FC_InitToken` call
- A plaintext secret or private key is zeroized when it is deleted with a `FC_DestroyObject` call

A standard C library function `PORT_memset`/`memset ( )` is used to zeroize memory used by plaintext secret and private keys and passwords. The `PORT_ZFree( )` function is used to free and zeroize the memory which is allocated from heap.

9.7 Random Number Generation

The NSS cryptographic module uses only the Approved RNG, implementing Algorithm Hash_DRBG of NIST SP 800-90 to generate cryptographic keys used by an approved security function. The certificate number of RNG obtained through the Cryptographic Algorithm Validation Program (CAVP) is #49.

10 Self-Tests (140-2 Section 4.9)

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

10.1 Power-Up Tests (140-2 Section 4.9.1)

The following tests are performed each time the NSS cryptographic module starts and must be completed successfully for the module to operate in the FIPS approved mode.

10.1.1 Cryptographic Algorithm Test

Table 12 Cryptographic Algorithm Tests (ref: fipstest.c, PKCS #11 FIPS Power-Up Self Test)

Algorithm	**Modes / Operation**	**Test**
Triple-DES	CBC (encrypt/decrypt)	KAT (Known Answer Test)
AES- 128, AES-192, AES-256,	ECB (encrypt/decrypt)	KAT
	CBC (encrypt/decrypt)	KAT
SHA-1, SHA-256, SHA-384, SHA-512	hash	KAT
HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	keyed hash (296 bit key)	KAT
RSA	encrypt/decrypt (1024-bit modulus *n*),	KAT
RSA-SHA-256/-SHA-384/-SHA-512	signature generation (2048-bit modulus *n*)	KAT
	signature verification (2048-bit modulus *n*)	KAT
DSA	key pair generation (1024-bit prime modulus *p*)	KAT
	signature generation (1024-bit prime modulus *p*)	KAT
	signature verification (1024-bit prime modulus *p*)	KAT
ECDSA	signature generation (Curve P-256; the Extend ECC version of the module also tests Curve K-283)	KAT
	signature verification (Curve P-256; the Extend ECC version of the module also tests Curve K-283)	KAT
RNG	N/A	KAT

Note: Cryptographic algorithms whose outputs vary for a given set of inputs (DSA and ECDSA) are tested using a known-answer test. The message digest algorithms have independent known-answer tests.

10.1.2 Software/Firmware Integrity Test

A software integrity test is performed on the libraries of the NSS cryptographic module. DSA is used as the approved authentication technique for the integrity test. If the test fails, then the module immediately enters the error state.

10.2 Conditional Tests (140-2 Section 4.9.2)

The following sections describe the conditional tests supported by the NSS cryptographic module.

10.2.1 Continuous Random Number Generator Test

The NSS cryptographic module performs a continuous random number generator test, whenever the pseudorandom number generator is invoked.

10.2.2 Pair-wise Consistency Test

The NSS cryptographic module performs a pair-wise consistency test, whenever the RSA, DSA, and/or ECDSA key pair generation is invoked.

10.2.3 Critical Function Test

No other critical function test is performed on power up.

11 Design Assurance (140-2 Section 4.10)

This section identifies the best practices used by Wind River Systems Inc during the design, deployment, and operation of the NSS cryptographic module.

11.1 Configuration management

The NSS cryptographic module is managed in a GIT-based configuration management system. The NSS module and its components within cryptographic boundary and associated module documentation are versioned, ensuring that changes and revisions to the modules can be controlled. Further, any and all changes to the source code are logged including the author of the change and the exact changes made.

11.2 Delivery and Operation

The NSS cryptographic modules are delivered as part of an overall root file-system in a single tar-ball. Within the tar-ball the modules are already configured and installed to start automatically by the operating system on boot. Further, the packages themselves within the root file-system are stripped and a checksum is computed to ensure that the packages have not been compromised during delivery. Upon delivery, the root file-system is simply installed directly onto a bare-metal target and booted. When booted, the system defaults to a secure mode of operation which ensures that no tampering may occur. In a secure mode of operation, all the security features (for example, process separation, access control etc.) required by a general purpose computer are enabled when the system is booted.

11.2.1 Security Rules

When in FIPS approved mode, the NSS Cryptographic Module, and all the applications using this module, must adhere to the following set of security rules:

The user must only use the FIPS 140-2 approved security functions listed in Section 2.4 or security functions allowed in the FIPS approved mode;

- The underlying operating system must ensure the integrity of the NSS cryptographic module loaded into memory;

  CAVEAT: The security assurances provided by the operating system are not a FIPS 140-2 requirement. Consequently, the module validation does not address if this is met or not met.
- All cryptographic keys used must be generated in the FIPS approved mode of operations or imported while running in the FIPS approved mode;
- The module controls the Critical Security Parameters (CSP) (for example, keys, and password, pin) and does not share the CSPs between an approved and a non-approved mode of operations. Secret and private keys are only to be passed to the calling application in encrypted (wrapped) form with `FC_WrapKey` using Triple-DES or AES (symmetric key algorithms) or RSA (asymmetric key algorithm). If a symmetric key algorithm is used to encrypt the secret and the private keys to pass to higher-level callers, the encryption key may derived from a password and, they should be considered to be in plaintext form in the FIPS approved mode;
- The software library binary of the module for each supported platform must only be installed on the corresponding platform listed in Section 2.2;
- In the FIPS approved mode, applications must call `FC_GetFunctionList` to obtain the function pointers providing the functional interface of the module. For all cryptographic operations, the applications must call the API function via these function pointers. Please note that when a `FC_Finalize`/`NSC_Initialize` sequence is executed, the module changes from FIPS approved mode to non-approved mode and when a `NSC_Finalize`/`FC_Initialize` sequence is executed; it changes from non-approved mode to FIPS approved mode. The operator may determine with the API function call FC_GETSlotInfo (slotDescription) whether the module is invoked for the Approved mode of operation.

- The environment variable NSS_ENABLE_AUDIT must be set to 1 before the application starts.
- The NSS cryptographic module must contain the following shared libraries/DLLs and the corresponding `.chk` files:

  `64- bit Wind River Secure Linux 1.0`

  - `/lib64/libfreebl3.chk`
  - `/lib64/libfreebl3.so`
  - `/lib64/libnssdbm3.chk`
  - `/lib64/libnssdbm3.so`
  - `/lib64/libsoftokn3.chk`
  - `/lib64/libsoftokn3.so`

The Wind River build system builds a complete pre-configured file-system and kernel that is delivered as a tar-ball and kernel bzImage. This is installed directly as a “bare metal” install onto the target. The preconfigured file-system contains all configuration files and other necessary files. When the target boots for the first time the system automatically generates checksums and performs self-tests on the NSS library to ensure that it is configured and running in FIPS approved mode. If these test fail and/or the NSS libraries are found to be running in non-FIPS mode the boot will log an error and enter a diagnostic/recovery mode of operation.

The libraries are installed in the directory /lib64.

The instructions to configure and build a file-system are as follows:

Install Wind River Secure Linux 1.0 DVD build environment

Configure a build specific to the intended target

Run “make” command within the target build

Install the resulting file-system tar-ball and kernel bzImage onto the target

Boot the target

Check to ensure the following libraries have read and execute privileges for everyone, and write access only to the owner (permission mask 0755):

- libsoftokn3.so
- libfreebl3.so Security_ Policy_Wind_River_NSS_3.12.4.pdf
- libnssdbm3.so

Also, ensure that the following checksum files have read access by all and write access by only the owner (permission mask 0644):

- libsoftokn3.chk
- libfreebl3.chk
- libnssdbm3.chk

11.3 Development

Please refer to chapter 2 for cryptographic module specification. The developer provided a well commented source code [NSS_CM3.12.4] of NSS Cryptographic Module.

11.4 Guidance Documents

The developer provided guidance in the form of Mozilla wiki pages. Please see to the references for a list of Mozilla wiki pages. Also, the guidance for user and crypto officer is provided in section 11.2

12 Mitigation of Other Attacks

The NSS cryptographic module is designed to mitigate the following attacks:

Table 13 Mitigation of Attacks

Other Attacks	Mitigation Mechanism	Specific Limitations
Timing attacks on RSA	Timing attack on RSA was first demonstrated by Paul Kocher in 1996 [TAIDHRD], who contributed the mitigation code to our module. Most recently Boneh and Brumley [RTAP] 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 [CMFP] 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 [PCCRS] recommend that every RSA signature generation should verify the signature just generated.	None

13 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

EMI/EMC properties of NSS cryptographic module are not meaningful for the library itself. Systems utilizing the NSS library services have their overall EMI/EMC ratings determined by the host system. The validation environments have FCC Class A and B ratings.

14 Sample Cryptographic Module Initialization Code

The following sample code uses NSPR functions (declared in the header file “prlink.h”) for dynamic library and function symbol lookup.

```
#include "prlink.h"
#include "cryptoki.h"
#include <assert.h>
#include <stdio.h>
#include <string.h>
/*
* An extension of the CK_C_INITIALIZE_ARGS structure for the * NSS cryptographic module. The
'LibraryParameters' field is
* used to pass instance-specific information to the library * (like where to find its config files, etc).
*/
typedef struct CK_C_INITIALIZE_ARGS_NSS {
            CK_CREATEMUTEX CreateMutex;
            CK_DESTROYMUTEX DestroyMutex;
            CK_LOCKMUTEX LockMutex;
            CK_UNLOCKMUTEX UnlockMutex;
            CK_FLAGS flags;
            CK_CHAR_PTR *LibraryParameters;
            CK_VOID_PTR pReserved;
}         CK_C_INITIALIZE_ARGS_NSS;

int main()
{
            char *libname;
            PRLibrary *lib;
            CK_C_GetFunctionList pFC_GetFunctionList;
            CK_FUNCTION_LIST_PTR pFunctionList;
            CK_RV rv;
            CK_C_INITIALIZE_ARGS_NSS initArgs;
            CK_SLOT_ID slotList[2], slotID;
            CK_ULONG ulSlotCount;
            CK_TOKEN_INFO tokenInfo;
            CK_SESSION_HANDLE hSession;
            CK_UTF8CHAR password[] = "1Mozilla";
            PRStatus status;


/*
*  Get the platform-dependent library name of the NSS
*  cryptographic module.
*/

            libname = PR_GetLibraryName(NULL, "softokn3");
            assert(libname!= NULL);
            lib = PR_LoadLibrary(libname);
            assert(lib!= NULL);
            PR_FreeLibraryName(libname);

pFC_GetFunctionList = (CK_C_GetFunctionList)

            PR_FindFunctionSymbol(lib, "FC_GetFunctionList");
            assert(pFC_GetFunctionList!= NULL);
            rv = (*pFC_GetFunctionList)(&pFunctionList);
assert(rv == CKR_OK);
```

```
/* Call FC_xxx via the function pointer pFunctionList->C_xxx */

initArgs.CreateMutex = NULL;
initArgs.DestroyMutex = NULL;
initArgs.LockMutex = NULL;
initArgs.UnlockMutex = NULL;
initArgs.flags = CKF_OS_LOCKING_OK;
initArgs.LibraryParameters = (CK_CHAR_PTR *)
            "configdir='.' certPrefix='' keyPrefix='' "
            "secmod='secmod.db' flags= ";
initArgs.pReserved = NULL;
rv = pFunctionList->C_Initialize(&initArgs);
assert(rv == CKR_OK);

ulSlotCount = sizeof(slotList)/sizeof(slotList[0]);
rv = pFunctionList->C_GetSlotList(CK_TRUE, slotList, &ulSlotCount);
assert(rv == CKR_OK);
slotID = slotList[0];

rv = pFunctionList->C_OpenSession(slotID,
CKF_RW_SESSION | CKF_SERIAL_SESSION, NULL, NULL, &hSession);
assert(rv == CKR_OK);

/* set the operator's initial password, if necessary */

rv = pFunctionList->C_GetTokenInfo(slotID, &tokenInfo);
assert(rv == CKR_OK);

if (!(tokenInfo.flags & CKF_USER_PIN_INITIALIZED)) {
/*
* As a formality required by the PKCS #11 standard, the
* operator must log in as the PKCS #11 Security Officer (SO),
* with the predefined empty string password, to set the
* operator's initial password.
*/
rv = pFunctionList->C_Login(hSession, CKU_SO, NULL, 0);
assert(rv == CKR_OK);

rv = pFunctionList->C_InitPIN(hSession,
password, strlen(password));
assert(rv == CKR_OK);

/* log out as the PKCS #11 SO */

rv = pFunctionList->C_Logout(hSession);
assert(rv == CKR_OK);
}
/* the module is now ready for use */

/* authenticate the operator using a password */

rv = pFunctionList->C_Login(hSession, CKU_USER,
password, strlen(password));
assert(rv == CKR_OK);
```

```
/* use the module's services ... */

rv = pFunctionList->C_CloseSession(hSession);
assert(rv == CKR_OK);

rv = pFunctionList->C_Finalize(NULL);
assert(rv == CKR_OK);

status = PR_UnloadLibrary(lib);
assert(status == PR_SUCCESS);
return 0;
}
```

The mode of operation of the NSS cryptographic module is determined by the second argument passed to the PR_FindFunctionSymbol function.

- For the non-FIPS approved mode of operation, look up the standard PKCS #11 function C_GetFunctionList
- For the FIPS approved mode of operation, look up the alternative function FC_GetFunctionList.