

FIPS PUB 140-1 Network Security Services Security Policy

[Updated to reflect NSS 3.2.2 Maintenance Validation]



1.1 Specification of Security Policy

A security policy includes the precise specification of the security rules under which the cryptographic module **must** operate, including rules derived from the security requirements of the FIPS PUB 140-1 standard, and the additional security rules listed below. The rules of operation of the cryptographic module that define within which role(s), and under what circumstances (when performing which services), an operator is allowed to maintain or disclose each security relevant data item of the cryptographic module.

There are three major reasons for developing and following a precise cryptographic module security policy:

- To induce the cryptographic module vendor (Sun Microsystems) to think carefully and precisely about who they want to access the cryptographic module, the way different system elements can be accessed, and which system elements to protect.
- To provide a precise specification of the cryptographic security to allow individuals and organizations (e.g., validators) to determine whether the cryptographic module, as implemented, does obey (satisfy) a stated security policy.
- To describe to the cryptographic module user (organization, or individual operator) the capabilities, protections, and access rights they will have when using the cryptographic module.

It should be noted that NSS utilizes RSA's PKCS #11, version 2.01, to form most of its cryptographic boundary. This, along with some certificate handling mechanisms, comprise the entire cryptographic module boundary. The following table states the various security policy rules which will be adhered to by each product utilizing NSS:

Table I. NSS Security Policy Rules

Rule	Statement of NSS Security Policy Rule
1	<i>The NSS cryptographic module shall consist of a series of binary software libraries compiled for each supported platform.</i>
2	<i>The cryptographic module shall rely on the underlying operating system to ensure the integrity of the cryptographic module loaded into memory.</i>

3	<i>The cryptographic module shall enforce a single role approach which is a combination of the User Role and the Cryptographic User Role as defined in FIPS PUB 140-1.</i>
4	<i>A cryptographic module user shall have access to ALL the services supplied by the cryptographic module.</i>
5	<i>Cryptographic module services shall consist of public services which require no authentication, and private services which require authentication.</i>
6	<i>Public key certificates shall be stored in plain text form because of their public nature and internal CA-signing integrity features.</i>
7	<i>SSL 2.0, 3.0, and TLS shall utilize authentication mechanisms above the cryptographic module which pass-through to utilize PKCS #11 authentication mechanisms which are within the cryptographic module.</i>
8	<i>SSL master secrets (private key data) shall be protected within the boundary of the cryptographic module (the SSL secure session ID cache shall be considered within the boundary of the cryptographic module).</i>
9	<i>For the FIPS PUB 140-1 mode of operation, the cryptographic module shall enforce rules specific to FIPS PUB 140-1 requirements.</i>
10	<i>The FIPS PUB 140-1 cryptographic module shall use an exception handling mechanism to ensure that critical errors are not allowed to compromise security (i. e. - whenever a critical error is encountered, the cryptographic module shall be required to be reinitialized).</i>
11	<p><i>Upon initialization of the FIPS PUB 140-1 cryptographic module, the following power-up self-tests shall be performed:</i></p> <ol style="list-style-type: none"> <i>(1) RC2-ECB Encrypt/Decrypt,</i> <i>(2) RC2-CBC Encrypt/Decrypt,</i> <i>(3) RC4 Encrypt/Decrypt,</i> <i>(4) DES-ECB Encrypt/Decrypt,</i> <i>(5) DES-CBC Encrypt/Decrypt,</i> <i>(6) triple DES-ECB Encrypt/Decrypt,</i> <i>(7) triple DES-CBC Encrypt/Decrypt,</i> <i>(8) MD2 Hash,</i> <i>(9) MD5 Hash,</i> <i>(10) SHA-1 Hash,</i> <i>(11) RSA Encrypt,</i> <i>(12) RSA Decrypt,</i> <i>(13) RSA Signature,</i> <i>(14) RSA Signature Verification,</i> <i>(15) DSA Signature, and</i> <i>(16) DSA Signature Verification.</i> <p><i>Additionally, if the user performs logout services, these same power-up self-tests are performed when the user logs back in to the FIPS PUB 140-1 cryptographic module.</i></p>
12	<i>Subsequent logins to the FIPS PUB 140-1 cryptographic module during the same established session shall execute the same series of power-up self-tests detailed above when logging-in under the FIPS PUB 140-1 mode. This allows a user to execute these power-up self-tests on</i>

	<i>demand as defined in section 4.11.1 of FIPS PUB 140-1.</i>
13	<i>The FIPS PUB 140-1 cryptographic module shall require the user to establish a password (for the user role) in order for subsequent authentications to be enforced.</i>
14	<i>All passwords shall be stored in an encrypted form in secondary storage.</i>
15	<i>Once a password has been established for the FIPS PUB 140-1 cryptographic module, it shall only allow the user to use security services if and only if the user successfully authenticates to the FIPS PUB 140-1 cryptographic module.</i>
16	<i>In order to verify the user's stored password, the user shall enter the password, and the verification that the password is correct shall be performed by the cryptographic module via PKCS #5 password-based encryption mechanisms.</i>
17	<i>The user's password shall act as the key material to encrypt/decrypt private key material.</i>
18	<i>The cryptographic module shall only extract private keys wrapped with a password according to PKCS #12.</i>
19	<i>Private keys, plain text PINs, and other security relevant data items (SRDIs) shall be maintained under the control of the cryptographic module, and shall not be passed to higher level callers.</i>
20	<i>All private keys shall be stored in an encrypted form in secondary storage.</i>
21	<i>Integrity checks shall be applied to the private and public key material retrieved from the database to ensure genuine data.</i>
22	<i>Once the FIPS PUB 140-1 mode of operation has been selected, the cryptographic module shall only allow FIPS PUB 140-1 cipher suite functionality.</i>
23	<i>The FIPS PUB 140-1 cipher suite shall consist solely of DES/Triple-DES (FIPS PUB 46-3) for encryption/decryption, SHA-1 (FIPS PUB 180-1) for hashing, RSA for key distribution, and DSA (FIPS PUB 186) or RSA (PKCS #1) for generic signature signing and verifying functionality.</i>
24	<i>Once the FIPS PUB 140-1 mode of operation has been selected, DES/Triple-DES shall be limited in its use to perform encryption/decryption using either CBC or ECB mode.</i>
25	<i>Once the FIPS PUB 140-1 mode of operation has been selected, SHA-1 shall be the only algorithm used to perform one-way hashes of data.</i>
26	<i>Once the FIPS PUB 140-1 mode of operation has been selected, RSA shall be limited in its use to generation of PKCS#1 signatures and verification of them, and to signing and verifying key material for key exchange.</i>
27	<i>Once the FIPS PUB 140-1 mode of operation has been selected, DSA shall be used in addition to RSA to generate signatures and to perform verification on them.</i>
28	<i>In the FIPS PUB 140-1 mode of operation, the cryptographic module shall perform a pairwise consistency test upon each invocation of RSA and DSA key generation as defined in section 4.11.2 of FIPS PUB 140-1.</i>
29	<i>The FIPS PUB 140-1 cryptographic module shall employ its prime number generation and verification via the mechanisms described in Appendix 2 of FIPS PUB 186.</i>
30	<i>The FIPS PUB 140-1 cryptographic module shall utilize pseudorandom number generation as defined via the mechanisms described in Appendix 3 of FIPS PUB 186.</i>

31	<i>The FIPS PUB 140-1 cryptographic module shall seed its pseudorandom number generation via invoking a noise generator specific to the platform on which it was implemented (e. g. - MacIntosh, UNIX, or Windows). Pseudorandom number generator shall be seeded with noise derived from the execution environment such that the noise is not predictable.</i>
32	<i>The FIPS PUB 140-1 cryptographic module's pseudorandom number generator shall periodically reseed itself with pseudorandom noise.</i>
33	<i>In the FIPS PUB 140-1 mode of operation, the cryptographic module shall perform a pseudorandom number generation test upon each invocation of the pseudorandom number generator as defined in section 4.11.2 of FIPS PUB 140-1.</i>
34	<i>Upon exit from the FIPS PUB 140-1 mode of operation, all security relevant data items within the cryptographic module which are stored to secondary storage shall be zeroized by having their memory contents rewritten with zeroes.</i>
35	<i>The TLS pseudorandom function (PRF) is contained within the cryptographic module, and it shall enforce if one hash is weak the PRF function would remain strong, this is accomplished by exclusive-oring the results of the two hashes in computation of security relevant data items -- specifically SSL pre-master secrets.</i>
36	<i>For operation in FIPS PUB 140-1 Level 2 mode, the machine shall be labeled in a tamper-evident manner. Labels are to be supplied by the vendor and placed by the user on the bottom right and left edges midway between the front and the back of the case. Before placing labels, clean the portion of the case where the labels will adhere with rubbing alcohol, and allow the case to dry. Apply the labels to the indicated locations, and allow labels to set for 24 hours.</i>
37	<i>The FIPS module is activated with a call to <code>SECMOD_DeleteModule()</code>, with the module to delete being the internal module. This will disable non-FIPS use of NSS, and enable the FIPS mode of operation. NSS clients may provide UI for enabling FIPS operation.</i>

Additionally, a cryptographic module security policy should be expressed in terms of the **roles**, **services**, **cryptographic keys**, and other **critical security parameters**. It should consist of, at a minimum, an **identification and authentication (I&A) policy** and an **access control policy**. An I&A policy specifies whether a cryptographic module operator is required to identify his or her self to the system, and, if so, what information is required and how it should be presented to the system in order for the operator to prove his or her identity to the system (i.e., authenticate themselves). Information required to be presented to the system might be passwords or individually unique biometric data. Once an operator can perform service(s) using the cryptographic module, an access control policy specifies what mode(s) of access he or she has to each security relevant data item while performing a given service.

1.2 Specification of Roles

A series of **security libraries** represent the cryptographic module which present the same application programmer interface (**API**) to client and server products utilizing NSS. There are minor variations, listed in the module interfaces description, but these do not break the following definition of roles. The NSS cryptographic module utilizes a single role approach -- this role is a combination of both the User Role and the Cryptographic Officer Role, and will be referenced below as **NSS User**. A NSS User utilizes secure services, and is also responsible for making decisions related to retrieval, updating, and deletion of keys from their key database. This is true for both client and server products. For multiple user products, like the HTTP

Server, the server still operates in this single role paradigm, under a single identity.

1.2.1 Authentication Policy

The NSS cryptographic module utilizes **Role-Based Authentication** - An operator who is allowed to use the cryptographic module must perform an authentication sequence using information unique to that operator (individual password) to perform sensitive services using the cryptographic module. Role-based authentication is utilized to safeguard a users **private key** information. However, Discretionary Access Controls (DAC) are used to safeguard all other NSS User information (e.g., the Public Key Certificate database). An NSS User may use a product (e.g. Netscape Navigator) without establishing a personal private key -- e.g., they may utilize SSL 3.0 Server Authentication without having a private key established. However, to enable SSL on the server products, a **private key** and **public key certificate** are required to enable secure services. An individual password is required in order to start the server -- this password is used to decrypt the private key.

1.3 Specification of Maintenance Roles

This section is not applicable to the NSS cryptographic module since it does not have a Maintenance Role.

1.4 Multiple Concurrent Operator Roles and Services

Since NSS-based applications always operate under a single role, under a single identity, no separate concurrent processes take place within an NSS-based application. In the case of separate threads of execution within the same process, NSS's threading model consists of a shared data segment with separate stack instances, and does not allow threads to leak insecurity into or out of the given process. Further, since a thread is not a separate process, and all threads of a given process live within the confines of that process, then all threads are subject to the same security imposed on the process itself.

1.5 Specification of Services

The vendor documentation shall fully describe each service including its purpose and function. Possible services may include, but not be limited to, the following:

- Cryptographic operations such as encryption, decryption, message integrity, digital signature generation, digital signature verification, and other operations that require the use of cryptography.
- Key management operations such as key and parameter entry, key generation, key output , key archiving, key zeroization, and other key management functions.
- Cryptographic management functions such as audit parameter entry and setting, alarm handling and resetting, and other cryptographic management functions.
- Performance of operator-selectable self tests, such as cryptographic algorithm tests, software/firmware tests , critical functions tests, statistical random number generator tests, or any additional tests that can be initiated by an operator.

The vendor documentation shall specify, **for each service**, the **service inputs**, corresponding **service outputs**, and the **authorized role** or roles in which the service can be performed. Service inputs shall consist of all data or control inputs to the module that initiate or obtain specific services, operations, or functions.

Service outputs shall consist of all data and status outputs that result from services, operations or functions initiated or obtained by service inputs. The vendor may supply a matrix that displays the services that can be performed in each role.

In each of the following services, since there is only one role, the user has access to **ALL** the services mediated by the application (for both client and server products). Routines have been specified for each service and denoted whether or not they are **public**, meaning that they require no authentication to utilize, or **private**, meaning that authentication must be provided prior to the routine being utilized. This model allows a type of **safety state** by allowing a NSS user to logout (thus disallowing any access to private services) without ending the session, and then log back in to re-authenticate private services rendered by the cryptographic module. All public and private services are listed in the following table:

Table II. Services

Name of Service	Description of Service in Terms of Routines
Certificate Storage and Retrieval	This private service consists of six routines used to perform certificate storage and retrieval including SEC_OpenPermCertDB(), AddCertToPermDB(), SEC_TraversePermCerts(), SEC_FindPermCertByKey(), SEC_DeletePermCertificate(), and CERT_ClosePermCertDB().
Digital Signatures	This private service consists of the one routine used to perform DSA signature generation, DSA_SignDigest(), and the one routine used to perform DSA signature verification, DSA_VerifyDigest(). This service also consists of the three routines used for RSA signature generation, verification, and entity association: RSA_Sign(), RSA_CheckSign(), and RSA_CheckSignRecover(), and the three raw routines used for RSA signature generation, verification, and entity association: RSA_SignRaw(), RSA_CheckSignRaw(), and RSA_CheckSignRecoverRaw(). In general, the key generation service must be invoked prior to invoking this service.
Encryption/Decryption	This private service consists of the four routines used to perform DES Encryption/Decryption including DES_CreateContext(), DES_Encrypt(), DES_Decrypt(), and DES_DestroyContext(). Single-key DES service is provided by using the NSS_DES and NSS_DES_CBC modes with DES_CreateContext(). Triple-DES service is provided by using the NSS_DES_EDE3 and NSS_DES_EDE3_CBC modes with DES_CreateContext().
Hashing	This public service consists of the eight routines used to perform SHA-1 hashing including SHA1_NewContext(), SHA1_CloneContext(), SHA1_Begin(), SHA1_Update(), SHA1_End(), SHA1_HashBuf(), SHA1_Hash(), and SHA1_DestroyContext().
Key Generation	This private service is utilized to perform key generation and consists of the six routines used to perform DSA key generation including PQGParamGen(), PQG_ParamGenSeedLen(), PQG_VerifyParams(), DSA_CreateKeyGenContext(), DSA_NewKey(), and DSA_NewKeyFromSeed(), and the one routine used for RSA private key generation called RSA_NewKey() (only used for entity association in public key exchange). When RSA_NewKey() is used in public key exchange between two parties, the Pairwise Consistency Test requires routines to check this symmetric algorithm. These consist of two routines used for entity association which include RSA_EncryptBlock(), and RSA_DecryptBlock(), and two raw routines used for entity association which include RSA_EncryptRaw(), and RSA_DecryptRaw().

<p>PKCS #5 Password-Based Encryption</p>	<p>The PKCS #5 API specifies a standard interface based upon the PKCS #5 standard which allows this private service to be used to perform password-based encryption and consists of the five routines including SEC_PKCS5GetSalt(), SEC_PKCS5GetIV(), SEC_PKCS5GetKey(), SEC_PKCS5CipherData(), and SEC_PKCS5CreateAlgorithmID().</p>																																																												
<p>PKCS #11</p>	<p>The PKCS #11 API specifies a standard interface based upon the PKCS #11 standard which allows for the selection of a FIPS PUB 140-1 mode of operation that provides both public and private services as well as a means of authentication into all private services, creates and maintains entry points for all FIPS PUB 140-1 specific routines including pk11_fipsPowerUpSelfTest() at initialization as well as on demand for subsequent logins, and enforces a pairwise consistency check on all key generation algorithms. NSS's FIPS PUB 140-1 PKCS #11 implementation defines the following standard crypto API:</p> <table border="1"> <thead> <tr> <th>Category</th> <th>Function</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>FIPS PUB 140-1 specific</td> <td>FC_GetFunctionList</td> <td>Return the list of FIPS PUB 140-1 functions</td> </tr> <tr> <td rowspan="3">General purpose</td> <td>FC_Initialize</td> <td>initializes Cryptoki</td> </tr> <tr> <td>FC_Finalize</td> <td>finalizes Cryptoki (1.1)</td> </tr> <tr> <td>FC_GetInfo</td> <td>obtains general information about Cryptoki</td> </tr> <tr> <td rowspan="8">Slot and token management</td> <td>FC_GetSlotList</td> <td>obtains a list of slots in the system</td> </tr> <tr> <td>FC_GetSlotInfo</td> <td>obtains information about a particular slot</td> </tr> <tr> <td>FC_GetTokenInfo</td> <td>obtains information about a particular token</td> </tr> <tr> <td>FC_GetMechanismList</td> <td>obtains a list of mechanisms supported by a token</td> </tr> <tr> <td>FC_GetMechanismInfo</td> <td>obtains information about a particular mechanism</td> </tr> <tr> <td>FC_InitToken</td> <td>initializes a token</td> </tr> <tr> <td>FC_InitPIN</td> <td>initializes the normal user's PIN</td> </tr> <tr> <td>FC_SetPIN</td> <td>modifies the PIN of the current user</td> </tr> <tr> <td rowspan="8">Session management</td> <td>FC_OpenSession</td> <td>opens a connection or "session" between an application and a particular token</td> </tr> <tr> <td>FC_CloseSession</td> <td>closes a session</td> </tr> <tr> <td>FC_CloseAllSessions</td> <td>closes all sessions with a token</td> </tr> <tr> <td>FC_GetSessionInfo</td> <td>obtains information about the session</td> </tr> <tr> <td>FC_GetOperationState</td> <td>saves the state of the cryptographic operation in a session (1.1)</td> </tr> <tr> <td>FC_SetOperationState</td> <td>restores the state of the cryptographic operation in a session (1.1)</td> </tr> <tr> <td>FC_Login</td> <td>logs into a token</td> </tr> <tr> <td>FC_Logout</td> <td>logs out from a token</td> </tr> <tr> <td rowspan="6">Object management</td> <td>FC_CreateObject</td> <td>creates an object</td> </tr> <tr> <td>FC_CopyObject</td> <td>creates a copy of an object</td> </tr> <tr> <td>FC_DestroyObject</td> <td>destroys an object</td> </tr> <tr> <td>FC_GetObjectSize</td> <td>obtains the size of an object in bytes</td> </tr> <tr> <td>FC_GetAttributeValue</td> <td>obtains an attribute value of an object</td> </tr> <tr> <td>FC_SetAttributeValue</td> <td>modifies an attribute value of an object</td> </tr> </tbody> </table>	Category	Function	Description	FIPS PUB 140-1 specific	FC_GetFunctionList	Return the list of FIPS PUB 140-1 functions	General purpose	FC_Initialize	initializes Cryptoki	FC_Finalize	finalizes Cryptoki (1.1)	FC_GetInfo	obtains general information about Cryptoki	Slot and token management	FC_GetSlotList	obtains a list of slots in the system	FC_GetSlotInfo	obtains information about a particular slot	FC_GetTokenInfo	obtains information about a particular token	FC_GetMechanismList	obtains a list of mechanisms supported by a token	FC_GetMechanismInfo	obtains information about a particular mechanism	FC_InitToken	initializes a token	FC_InitPIN	initializes the normal user's PIN	FC_SetPIN	modifies the PIN of the current user	Session management	FC_OpenSession	opens a connection or "session" between an application and a particular token	FC_CloseSession	closes a session	FC_CloseAllSessions	closes all sessions with a token	FC_GetSessionInfo	obtains information about the session	FC_GetOperationState	saves the state of the cryptographic operation in a session (1.1)	FC_SetOperationState	restores the state of the cryptographic operation in a session (1.1)	FC_Login	logs into a token	FC_Logout	logs out from a token	Object management	FC_CreateObject	creates an object	FC_CopyObject	creates a copy of an object	FC_DestroyObject	destroys an object	FC_GetObjectSize	obtains the size of an object in bytes	FC_GetAttributeValue	obtains an attribute value of an object	FC_SetAttributeValue	modifies an attribute value of an object
Category	Function	Description																																																											
FIPS PUB 140-1 specific	FC_GetFunctionList	Return the list of FIPS PUB 140-1 functions																																																											
General purpose	FC_Initialize	initializes Cryptoki																																																											
	FC_Finalize	finalizes Cryptoki (1.1)																																																											
	FC_GetInfo	obtains general information about Cryptoki																																																											
Slot and token management	FC_GetSlotList	obtains a list of slots in the system																																																											
	FC_GetSlotInfo	obtains information about a particular slot																																																											
	FC_GetTokenInfo	obtains information about a particular token																																																											
	FC_GetMechanismList	obtains a list of mechanisms supported by a token																																																											
	FC_GetMechanismInfo	obtains information about a particular mechanism																																																											
	FC_InitToken	initializes a token																																																											
	FC_InitPIN	initializes the normal user's PIN																																																											
	FC_SetPIN	modifies the PIN of the current user																																																											
Session management	FC_OpenSession	opens a connection or "session" between an application and a particular token																																																											
	FC_CloseSession	closes a session																																																											
	FC_CloseAllSessions	closes all sessions with a token																																																											
	FC_GetSessionInfo	obtains information about the session																																																											
	FC_GetOperationState	saves the state of the cryptographic operation in a session (1.1)																																																											
	FC_SetOperationState	restores the state of the cryptographic operation in a session (1.1)																																																											
	FC_Login	logs into a token																																																											
	FC_Logout	logs out from a token																																																											
Object management	FC_CreateObject	creates an object																																																											
	FC_CopyObject	creates a copy of an object																																																											
	FC_DestroyObject	destroys an object																																																											
	FC_GetObjectSize	obtains the size of an object in bytes																																																											
	FC_GetAttributeValue	obtains an attribute value of an object																																																											
	FC_SetAttributeValue	modifies an attribute value of an object																																																											

	FC_FindObjectsInit	initializes an object search operation
	FC_FindObjects	continues an object search operation
	FC_FindObjectsFinal	finishes an object search operation (1.1)
Encryption and decryption	FC_EncryptInit	initializes an encryption operation
	FC_Encrypt	encrypts single-part data
	FC_EncryptUpdate	continues a multiple-part encryption operation
	FC_EncryptFinal	finishes a multiple-part encryption operation
	FC_DecryptInit	initializes a decryption operation
	FC_Decrypt	decrypts single-part encrypted data
	FC_DecryptUpdate	continues a multiple-part decryption operation
	FC_DecryptFinal	finishes a multiple-part decryption operation
	Message digesting	FC_DigestInit
FC_Digest		digests single-part data
FC_DigestUpdate		continues a multiple-part digesting operation
FC_DigestKey		continues a multi-part message-digesting operation by digesting the value of a secret key as part of the data already digested (1.1)
FC_DigestFinal		finishes a multiple-part digesting operation
Signature and verification	FC_SignInit	initializes a signature operation
	FC_Sign	signs single-part data
	FC_SignUpdate	continues a multiple-part signature operation
	FC_SignFinal	finishes a multiple-part signature operation
	FC_SignRecoverInit	initializes a signature operation, where the data can be recovered from the signature
	FC_SignRecover	signs single-part data, where the data can be recovered from the signature
	FC_VerifyInit	initializes a verification operation
	FC_Verify	verifies a signature on single-part data
	FC_VerifyUpdate	continues a multiple-part verification operation
	FC_VerifyFinal	finishes a multiple-part verification operation
	FC_VerifyRecoverInit	initializes a verification operation where the data is recovered from the signature
	FC_VerifyRecover	verifies a signature on single-part data, where the data is recovered from the signature
Dual-function cryptographic operations	FC_DigestEncryptUpdate	continues a multiple-part digesting and encryption operation (1.1)
	FC_DecryptDigestUpdate	continues a multiple-part decryption and digesting operation (1.1)
	FC_SignEncryptUpdate	continues a multiple-part signing and encryption operation (1.1)
	FC_DecryptVerifyUpdate	continues a multiple-part decryption and verify operation (1.1)
Key management	FC_GenerateKey	generates a secret key
	FC_GenerateKeyPair	generates a public-key/private-key pair
	FC_WrapKey	wraps (encrypts) a key

	FC_UnwrapKey	unwraps (decrypts) a key
	FC_DeriveKey	derives a key from a base key
Random number generation	FC_SeedRandom	mixes in additional seed material to the random number generator
	FC_GenerateRandom	generates random data
Function management	FC_GetFunctionStatus	obtains updated status of a function running in parallel with the application
	FC_CancelFunction	Cancels a function running in parallel with the application
Callbacks	Notify	processes notifications from Cryptoki
PKCS #12 Personal Information Exchange	The PKCS #12 API will specify a standard interface based upon the forthcoming PKCS #12 standard which allows this private service to be used to exchange data such as private keys and certificates between two parties and consists of the eight routines including SEC_PKCS12CreateExportContext, SEC_PKCS12CreatePasswordPrivSafe(), SEC_PKCS12AddCertAndKey(), SEC_PKCS12Encode(), SEC_PKCS12DestroyExportContext(), SEC_PKCS12DecoderStart(), SEC_PKCS12DecoderUpdate(), and SEC_PKCS12DecoderFinish().	
Prime Number Generation	This public service consists of the two routines used for generating a prime number including mpp_make_prime() and mpp_pprime().	
Private Key Storage and Retrieval	This private service is utilized to perform private key storage and retrieval and consists of the seven routines including SECKEY_OpenKeyDB(), SECKEY_TraverseKeys(), SECKEY_UpdateKeyDBPass1() SECKEY_UpdateKeyDBPass2(), SECKEY_FindKeyByPublicKey(), SECKEY_DeleteKey(), and SECKEY_CloseKeyDB().	
Pseudorandom Number Generation	This public service consists of the four routines used for global pseudorandom number generation including RNG_RNGInit(), RNG_GenerateGlobalRandomBytes(), RNG_RandomUpdate(), and RNG_RNGShutdown(), and the three routines used for seeding pseudorandom number generation including RNG_GetNoise(), RNG_SystemInfoForRNG(), and RNG_FileForRNG(). A continuous pseudorandom number generator test is performed whenever a new pseudorandom number is generated.	
SSL Session ID Cache (Secret Management)	This public service consists of the five routines used to perform session ID cache management including SSL_ConfigServerSessionIDCache(), ssl_FreeSID(), ssl_LookupSID(), ssl_ChooseSessionIDProcs(), and SSL_ClearSessionCache().	
TLS pseudorandom function (PRF)	TLS pseudorandom function (PRF) is utilized by SSL 3.0 protocol to produce FIPS 140-1 compliant hashes of security relevant data items [pre-master secret]. See SSL changes in Security Module 1.01 for full details.	

1.6 Bypass Capabilities

This section is not applicable to NSS since it does not allow for any bypass capability.

1.7 Access Control Policy

The access control policy enforced by the cryptographic module must be sufficiently precise, and of sufficient detail to allow the operator and testers to know what **security relevant data items** the **operator** has access to while performing a **service**, and the **modes** of access he or she has to these data items. Also, the testers and operator must be able to know if and how the kinds of data items accessible changes when the service is invoked from each role in which it can be invoked.

1.7.1 Security Relevant Data Items

Security relevant data items consist of data types used for Certificate Storage and Retrieval, Digital Signatures, Encryption/Decryption, Generic Containers, Hashing, Key Generation, PKCS #5 Password-Based Encryption, PKCS #12 Personal Information Exchange, Private Key Storage and Retrieval, Pseudorandom Number Generation, and SSL Session ID Cache (Secret Management).

All security relevant data items are identified by category, type, name, and description in the following table:

Table III. Security Relevant Data Items

Category	Type of Data Item	Name of Data Item	Description of Data Item
Generic Containers	typedef struct SECAgorithmIDStr	SECAgorithmID	The structure containing two SECItems which identify the X.500 algorithm.
	typedef struct SECItemStr	SECItem	Generic container used to hold type of data, actual data content, and length of data.
	typedef struct SECKEYLowPrivateKeyStr	SECKEYLowPrivateKey	Generic container used for low-level private key structures including RSA and DSA private keys. This structure is used below the PKCS #11 service layer and contains the actual private key.
	typedef struct SECKEYLowPublicKeyStr	SECKEYLowPublicKey	Generic container used for low-level public key structures including RSA and DSA public keys. This structure is used below the PKCS #11 service layer and contains the actual public key.
	typedef struct SECKEYPrivateKeyStr	SECKEYPrivateKey	Generic container used as a high-level pointer to the defined private key structures, and is used above the PKCS #11 service layer.
	typedef struct SECKEYPublicKeyStr	SECKEYPublicKey	Generic container used as a high-level pointer to the defined public key structures, and is used above the

			PKCS #11 service layer.
	typedef enum	SECOidTag	Generic container used to identify the supported object IDs.
	typedef enum _SECStatus	SECStatus	Generic container used primarily to indicate success or failure.
Certificate Storage and Retrieval	typedef struct CERTCertificateStr	CERTCertificate	The structure representing an X.509 certificate object (the unsigned form).
	typedef struct CERTCertDBHandleStr	CERTCertDBHandle	The structure representing a handle to an open certificate database.
	typedef struct CERTCertTrustStr	CERTCertTrust	The trust structure containing flags for SSL and email.
	typedef struct _certDBEntryCert	certDBEntryCert	The structure for certificate database entries.
Digital Signatures	typedef struct DSAPrivateKeyStr	DSAPrivateKey	The structure representing the context of a digital signature containing data associated with the private portion of the DSA key pair.
	typedef struct DSAPublicKeyStr	DSAPublicKey	The structure representing the context of a digital signature verification containing data associated with the public portion of the DSA key pair.
	typedef struct RSAPrivateKeyStr	RSAPrivateKey	The structure representing the context of an RSA signature generation

			or decryption mechanism used for both signature generation and key exchange; containing data associated with the private portion of the RSA key pair.
	typedef struct RSAPublicKeyStr	RSAPublicKey	The structure representing the context of an RSA signature verification or encryption mechanism used for both signature verification and key exchange; containing data associated with the public portion of the RSA key pair.
Encryption/ Decryption	typedef struct DESContextStr	DESContext	The structure representing the context of a DES encryption/decryption containing an encrypt/decrypt flag, space for up to three distinct keys, space for the carry-forward needed for CBC modes of DES, and function pointers to the appropriate encryption and decryption functions associated with that mode of DES.
Hashing	typedef struct SHA1ContextStr	SHA1Context	The structure representing the context of a SHA-1 hash containing information relevant to performing a SHA-1 hash.

Key Generation	typedef struct PQGParamsStr	PQGParams	The structure representing the context of a digital signature key generation containing multiple items including pointers to both low-level public and private key structures containing the public and private portions of the DSA key pair.
	typedef struct PQGVerifyStr	PQGVerify	The structure representing the context of a digital signature containing data associated with the verification (in terms of validity) of a set of parameters contained in a DSA key pair.
	typedef struct DSAPrivateKeyStr	DSAPrivateKey	The structure containing the private portion of the DSA key pair.
	typedef struct DSAPublicKeyStr	DSAPublicKey	The structure containing the public portion of the DSA key pair.
	typedef struct RSAPrivateKeyStr	RSAPrivateKey	The structure containing the private portion of the RSA key pair.
	typedef struct RSAPublicKeyStr	RSAPublicKey	The structure containing the public portion of the RSA key pair.
PKCS #5 Password-Based Encryption	typedef struct SECItemStr	SECItem	Utilizes this generic container to hold password-based encryption data.

PKCS #12 Personal Information Exchange	typedef struct SEC_PKCS12ExportContextStr	SEC_PKCS12ExportContext	The structure representing the context of a PKCS#12 export operation.
	typedef struct SEC_PKCS12DecoderContextStr	SEC_PKCS12DecoderContext	The structure representing the context of a PKCS#12 import operation.
Prime Number Generation	typedef struct	mp_int	The structure used to hold very large numbers.
	typedef int	mp_err	The integer used to hold error codes from the Multi-Precision Arithmetic (big integer) library.
Private Key Storage and Retrieval	typedef struct SECKEYKeyDBHandleStr	SECKEYKeyDBHandle	The structure representing a handle into the private key database.
	typedef struct SECKEYLowPrivateKeyStr	SECKEYLowPrivateKey	Utilizes this generic container used for low-level private key structures.
SSL Session ID Cache (Secret Management)	typedef struct SSLSecurityInfoStr	SSLSecurityInfo	The structure containing all information relevant to SSL security.
	typedef struct SSLSessionIDStr	SSLSessionID	The structure containing data relevant to the SSL session ID including the session ID cache and the master secret.

1.7.2 Service Relationships to Security Relevant Data Items Matrix

Table IV. Service Routine to Security Relevant Data Items Matrix

Service	Service Routine	Security Relevant Data Item	Read Access	Write Access	
Certificate Storage and Retrieval	AddCertToPermDB()	CERTCertDBHandle	X	X	
		CERTCertificate	X	X	
		CERTCertTrust	X	X	
		certDBEntryCert	X	-	
	CERT_ClosePermCertDB()	CERTCertDBHandle	X	X	
	SEC_FindPermCertByKey()	CERTCertDBHandle	X	X	
		SECItem	X	X	
		certDBEntryCert	X	-	
	SEC_OpenPermCertDB()	CERTCertDBHandle	X	X	
		SECStatus	X	-	
	SEC_DeletePermCertificate()	CERTCertDBHandle	X	X	
		CERTCertificate	X	X	
		SECStatus	X	-	
	SEC_TraversePermCerts()	CERTCertDBHandle	X	X	
SECStatus		X	-		
Digital Signatures	DSA_SignDigest()	DSAPrivateKey	X	-	
		SECStatus	X	-	
	DSA_VerifyDigest()	DSAPublicKey	X	-	
		SECStatus	X	-	
	RSA_Sign()	SECKEYLowPrivateKey	X	-	
		SECStatus	X	-	
	RSA_CheckSign()	SECKEYLowPublicKey	X	-	
		SECStatus	X	-	
	RSA_CheckSignRecover()	SECKEYLowPublicKey	X	-	
		SECStatus	X	-	
	RSA_SignRaw()	SECKEYLowPrivateKey	X	-	
		SECStatus	X	-	
	RSA_CheckSignRaw()	SECKEYLowPublicKey	X	-	
		SECStatus	X	-	
	RSA_CheckSignRecoverRaw()	SECKEYLowPublicKey	X	-	
		SECStatus	X	-	
	Encryption/Decryption	DES_CreateContext()	DESContext	-	X
		DES_Encrypt()	DESContext	X	X
SECStatus			X	-	

	DES_Decrypt()	DESContext	X	X
		SECStatus	X	-
	DES_DestroyContext()	DESContext	-	X
Hashing	SHA1_NewContext()	SHA1Context	-	X
	SHA1_CloneContext()	SHA1Context	X	-
		SHA1Context	-	X
	SHA1_Begin()	SHA1Context	-	X
	SHA1_Update()	SHA1Context	X	X
	SHA1_End()	SHA1Context	X	X
	SHA1_HashBuf()	SECStatus	X	-
	SHA1_Hash()	SECStatus	X	-
SHA1_DestroyContext()	SHA1Context	-	X	
Key Generation	PQG_ParamGen()	PQGParams	-	X
		PQGVerify	-	X
		SECStatus	X	-
	PQG_ParamGenSeedLen()	PQGParams	-	X
		PQGVerify	-	X
		SECStatus	X	-
	PQG_VerifyParams()	PQGParams	X	-
		PQGVerify	X	-
		SECStatus	X	-
	DSA_NewKey()	PQGParams	X	-
		DSAPrivateKey	-	X
		SECStatus	X	-
	DSA_NewKeyFromSeed()	PQGParams	X	-
		DSAPrivateKey	-	X
		SECStatus	X	-
	RSA_NewKey()	RSAPrivateKey	-	X
	RSA_EncryptBlock()	SECKEYLowPublicKey	X	-
		SECStatus	X	-
	RSA_DecryptBlock()	SECKEYLowPrivateKey	X	-
		SECStatus	X	-
RSA_EncryptRaw()	SECKEYLowPublicKey	X	-	
	SECStatus	X	-	
RSA_DecryptRaw()	SECKEYLowPrivateKey	X	-	
	SECStatus	X	-	

PKCS #5 Password-Based Encryption	SEC_PKCS5GetSalt()	SECAgorithmID	X	X
		SECItem	X	-
	SEC_PKCS5GetIV()	SECAgorithmID	X	X
		SECItem	X	-
	SEC_PKCS5GetKey()	SECAgorithmID	X	X
		SECItem	X	-
	SEC_PKCS5CipherData()	SECAgorithmID	X	X
		SECItem	X	X
		SECItem	X	-
	SEC_PKCS5CreateAlgorithmID()	SECOidTag	X	-
		SECItem	X	X
		SECAgorithmID	-	X
PKCS #12 Personal Information Exchange	SEC_PKCS12CreateExportContext()	SEC_PKCS12ExportContext	X	X
	SEC_PKCS12CreatePasswordPrivSafe()	SEC_PKCS12ExportContext	X	-
	SEC_PKCS12AddCertAndKey()	SEC_PKCS12ExportContext	X	-
		CERTCertificate	X	-
		SECStatus	X	-
	SEC_PKCS12Encode()	SEC_PKCS12ExportContext	X	-
		SECStatus	X	-
	SEC_PKCS12DestroyExportContext()	SEC_PKCS12ExportContext	-	X
	SEC_PKCS12DecoderStart()	SEC_PKCS12DecoderContext	X	X
	SEC_PKCS12DecoderUpdate()	SEC_PKCS12DecoderContext	X	-
SECStatus		X	-	
SEC_PKCS12DecoderFinish()	SEC_PKCS12DecoderContext	-	X	
Prime Number Generation	mpp_make_prime()	mp_int	X	X
		mp_err	X	-
	mpp_pprime()	mp_int	X	-
		mp_err	X	-
Private Key Storage and Retrieval	SECKEY_CloseKeyDB()	SECKEYKeyDBHandle	X	X
	SECKEY_DeleteKey()	SECKEYKeyDBHandle	X	X
		CERTCertificate	X	-
	SECKEY_FindKeyByCert()	SECKEYKeyDBHandle	X	X
		CERTCertificate	X	X
		SECKEYLowPrivateKey	X	X
SECKEY_OpenKeyDB()	SECKEYKeyDBHandle	X	-	

	SECKEY_TraversePermKeys()	SECKEYKeyDBHandle	X	X
		SECStatus	X	-
	SECKEY_UpdateKeyDBPass1()	SECKEYKeyDBHandle	X	X
		SECStatus	X	-
	SECKEY_UpdateKeyDBPass2()	SECKEYKeyDBHandle	X	X
		SECItem	X	X
		SECStatus	X	-
Pseudorandom Number Generation	RNG_RNGInit()	SECStatus	X	-
	RNG_GenerateGlobalRandomBytes()	SECStatus	X	-
	RNG_RandomUpdate()	SECStatus	X	-
	RNG_RNGShutdown()	void	-	-
SSL Session ID Cache (Secret Management)	ssl_ChooseSessionIDProcs()	SSLSecurityInfo	X	X
		SSLSessionID	-	X
	SSL_ClearSessionCache()	SSLSessionID	X	X
	ssl_LookupSID()	SSLSessionID	X	X
		SSLSessionID	X	-
	ssl_FreeSID()	SSLSessionID	X	X
		SSLSessionID	-	X
SSL pre-master secrets	pk11_PRF()	const SECItem *secret	X	X

1.8 Means of Access

Prior to execution of the Client or Server products, the Security Libraries are stored on disk in compiled binary form. NSS *relies on Discretionary Access Controls (DAC)* to protect the binary image from being tampered with.

1.9 Zeroization

Within the Security Libraries, there are a number of explicit zeroization steps that are taken to clear the memory region previously occupied by a private key or password. In summary, private keys are not stored in plaintext. Any key material that has been unwrapped for use is zeroed once the use is complete. The function used to both zero and free memory used by private key material is PORT_ZFree().

1.10 Role-based Authentication

Since all NSS-based products utilize role-based authentication, and all products use a single-role mechanism referred to above as a NSS User, authentication shall always be required upon initializing the FIPS Cryptographic Module. This is true of all NSS-based client and server products, and shall be handled via the PKCS #11 mechanism of required authentication.

1.11 Identity-based Authentication

This section is not applicable to NSS since it is only applicable to products attempting to be certified to security level three or four.

Results of FIPS 140-1 Level 2 Maintenance Validation of NSS 3.2.2

FIPS 140-1 Section	Description	Validation Level Obtained
1.0	Cryptographic Modules	2
2.0	Module Interfaces	2
3.0	Roles and Services	2
4.0	Finite State Machine Model	2
5.0	Physical Security	2
6.0	Software Security	2
7.0	Operating System Security	2
8.0	Cryptographic Key Management	2
9.0	Cryptographic Algorithms	2
10.0	EMI/EMC	2
11.0	Self-Tests	2

Results of FIPS 140-1 Level 1 Maintenance Validation of NSS 3.2.2

FIPS 140-1 Section	Description	Validation Level Obtained
1.0	Cryptographic Modules	1
2.0	Module Interfaces	1
3.0	Roles and Services	2
4.0	Finite State Machine Model	1
5.0	Physical Security	1

6.0	Software Security	1
7.0	Operating System Security	1
8.0	Cryptographic Key Management	1
9.0	Cryptographic Algorithms	1
10.0	EMI/EMC	1
11.0	Self-Tests	1

Platform List

To meet the FIPS 140-1 level 1 requirement, the operating system on which NSS runs must allow only one user at a time.

Windows 95, 98, and Me are single-user operating systems. Other operating systems (Windows NT 4.0, Windows 2000, SunOS, Linux, AIX, HP-UX, and OSF1) must be running in single-user mode.

For the level 2 certificate, the platform validated was a Sun Ultra 1 running Solaris 8 and was configured according to the specifications listed in the Common Criteria documents referred to in the link below

<http://www.commoncriteria.org/ccc/epl/productType/epldetail.jsp?id=42>

Platform Validated	Level Obtained
Windows 98	1
SunOS 5.8	2

Vendor Affirmed Platform	Level
Windows 95, Me, NT 4.0, 2000	1
SunOS 5.6 SPARC SunOS 5.8 SPARC 32-bit, 64-bit SunOS 5.8 x86	1
Linux 2.2	1
AIX 4.3.3 32-bit, 64-bit	1
HP-UX B.11.00 32-bit, 64-bit	1
OSF1 V5.0A	1