FIPS PUB 140-1
Netscape Security Policy

Updated 3/15/99 to reflect NIST / Infogard recommended changes

[Updated to reflect Security Module 1.01 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 imposed by **Netscape**. 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 (Netscape) 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 Netscape utilizes RSA's PKCS #11, version 1.1, 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 Netscape product:

Table I. Netscape Security Policy Rules

Rule	Statement of Netscape Security Policy Rule
1	*Netscape's cryptographic module shall consist of a series of binary software libraries compiled for each supported platform and utilized by ALL Netscape client and server products.*

2	*The cryptographic module shall rely on the underlying operating system to ensure the integrity of the cryptographic module loaded into memory.*
3	*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.*
4	*A cryptographic module user shall have access to ALL the services supplied by the cryptographic module.*
5	*Cryptographic module services shall consist of public services which require no authentication, and private services which require authentication.*
6	*Public key certificates shall be stored in plain text form because of their public nature and internal CA-signing integrity features.*
7	*SSL 2.0 and 3.0 shall utilize authentication mechanisms above the cryptographic module which pass-through to utilize PKCS #11 authentication mechanisms which are within the cryptographic module.*
8	*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).*
9	*For the FIPS PUB 140-1 mode of operation, the cryptographic module shall enforce rules specific to FIPS PUB 140-1 requirements.*
10	*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).*
11	*Upon initialization of the FIPS PUB 140-1 cryptographic module, the following power-up self-tests shall be performed:* *(1) RC2-ECB Encrypt/Decrypt,* *(2) RC2-CBC Encrypt/Decrypt,* *(3) RC4 Encrypt/Decrypt,* *(4) DES-ECB Encrypt/Decrypt,* *(5) DES-CBC Encrypt/Decrypt,* *(6) triple DES-ECB Encrypt/Decrypt,* *(7) triple DES-CBC Encrypt/Decrypt,* *(8) MD2 Hash,* *(9) MD5 Hash,* *(10) SHA-1 Hash,* *(11) RSA Encrypt,* *(12) RSA Decrypt,* *(13) RSA Signature,* *(14) RSA Signature Verification,* *(15) DSA Signature, and* *(16) DSA Signature Verification.* *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.
12	*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 demand as defined in section 4.11.1 of FIPS PUB 140-1.*
13	*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.*
14	*All passwords shall be stored in an encrypted form in secondary storage.*
15	*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.*
16	*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.*
17	*The user's password shall act as the key material to encrypt/decrypt private key material via PKCS #5 using Triple-DES.*
18	*The cryptographic module shall only extract private keys wrapped with a password according to PKCS #12.*
19	*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.*
20	*All private keys shall be stored in an encrypted form in secondary storage.*
21	*Integrity checks shall be applied to the private and public key material retrieved from the database to ensure genuine data.*
22	*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.*
23	*The FIPS PUB 140-1 cipher suite shall consist solely of DES (FIPS PUB 46-2) for encryption/decryption, SHA-1 (FIPS PUB 180-1) for hashing, RSA for key distribution, and DSA (FIPS PUB 186) for generic signature signing and verifying functionality.*
24	*Once the FIPS PUB 140-1 mode of operation has been selected, DES and triple-DES shall be limited in its use to perform encryption/decryption using either CBC (TCBC) or ECB (TECB) mode.*
25	*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.*
26	*Once the FIPS PUB 140-1 mode of operation has been selected, RSA can be used for signature functionality to sign and verify key material for key exchange and perform general purpose signatures.*
27	*Once the FIPS PUB 140-1 mode of operation has been selected, DSA can be used to generate signatures and perform verification on them for general purpose signatures.*
28	*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.
29	*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.*
30	*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.*
31	*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.*
32	*The FIPS PUB 140-1 cryptographic module's pseudorandom number generator shall periodically reseed itself with pseudorandom noise.*
33	*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.*
34	*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.*
35	*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.*

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 all Netscape client and server products. There are minor variations, listed in the module interfaces description, but these do not break the following definition of roles. Netscape's 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 **Netscape User**. A Netscape 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 (Enterprise Server 3.0), the server still operates in this single role paradigm, under a single identity.

1.2.1 Authentication Policy

Netscape's 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 Netscape User information (e.g., the Public Key Certificate database). A Netscape 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 Netscape products since they do not have a Maintenance Role.

1.4 Multiple Concurrent Operator Roles and Services

Since Netscape applications always operate under a single role, under a single identity, no separate concurrent processes take place within a Netscape application. In the case of separate threads of execution within the same process, Netscape'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 Netscape 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 **four** routines used to perform DSA signature generation including DSA_CreateSignContext(), DSA_PreSign(), DSA_Sign(), and DSA_DestroySignContext(), and the **three** routines used to perform DSA signature verification including DSA_CreateVerifyContext(), DSA_Verify(), and DSA_DestroyVerifyContext(). Performing public key exchange between two parties or performing RSA signature generation, consists of the **three** routines used for entity association, or performing RSA signature generation, including RSA_Sign(), RSA_CheckSign(), and RSA_CheckSignRecover(), and the **three** raw routines used for entity association including RSA_SignRaw(), RSA_CheckSignRaw(), and RSA_CheckSignRecoverRaw(). In general, the key generation service must be invoked prior to invoking this service.
	This **private** service consists of the **five** routines used to perform DES or

Encryption/ Decryption	triple-DES Encryption/Decryption including DES_CreateContext(), DES_Encrypt(), DES_Decrypt(), DES_PadBuffer(), and DES_DestroyContext().
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 **three** routines used to perform DSA key generation including DSA_CreateKeyGenContext(), DSA_KeyGen(), and DSA_DestroyKeyGenContext(), and the **one** routine used for RSA private key generation called RSA_NewKey(). 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 which include RSA_EncryptBlock(), and RSA_DecryptBlock(), and **two** raw routines which include RSA_EncryptRaw(), and RSA_DecryptRaw().
PKCS #5 Password-Based Encryption	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 **three** routines including SEC_PKCS5GetSalt(), SEC_PKCS5CipherData(), and SEC_PKCS5CreateAlgorithmID().
	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. Netscape's FIPS PUB 140-1 PKCS #11 implementation defines the following standard crypto API:

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_GetMechansimList	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
PKCS #11		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	initializes a message-digesting operation
	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	FC_GenerateKey	generates a secret key

	management	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 **two** routines including SEC_PKCS12GetPFX() and SEC_PKCS12PutPFX().		
Prime Number Generation	This **public** service consists of the **four** routines used for generating a prime number including prm_PrimeFind(), prm_GeneratePrimeRoster(), prm_PseudoPrime(), and prm_RabinTest().		
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_ResetRandom(), the **six** routines used for pseudorandom number generation on a per object basis including RNG_CreateContext(), RNG_Init(), RNG_GenerateRandomBytes(), RNG_Update(), RNG_Reseed(), and RNG_DestroyContext(), 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 Netscape products since they do 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
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 DSASignContextStr	DSASignContext	The structure representing the context of a digital signature containing data associated with the private portion of the DSA key pair.
	typedef struct DSAVerifyContextStr	DSAVerifyContext	The structure representing the context of a digital signature verification containing data associated with the public portion of the DSA key pair.
	typedef struct RSAPrivateContextStr	RSAPrivateContext	The structure representing the context of an RSA signature generation or decryption mechanism containing data associated with the private portion of the RSA key pair.
	typedef struct RSAPublicContextStr	RSAPublicContext	The structure representing the context of an RSA signature verification or encryption mechanism 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 or triple-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.
Generic Containers	typedef struct	CMPInt	Generic container used to hold very large numbers.
	typedef struct SECAlgorithmIDStr	SECAlgorithmID	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.
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 DSAKeyGenContextStr	DSAKeyGenContext	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 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 RSAKeyGenContextStr	RSAKeyGenContext	The structure representing the context of a key generation used for key exchange containing multiple items including a low-level private key structure containing the private portion of the RSA key pair (and the public portion of the RSA key pair which is replicated inside of the private portion of the RSA 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 SECItemStr	SECItem	Utilizes this generic container for data associated with personal information exchange.
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.
Pseudorandom Number Generation	typedef struct RNGContextStr	RNGContext	The structure representing the context of pseudorandom number generation dependent upon a SHA1Context and a seed value among other data items.
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_CreateSignContext()	SECKEYLowPrivateKey	X	-
		DSASignContext	-	X
	DSA_PreSign()	DSASignContext	X	X
		SECStatus	X	-
	DSA_Sign()	DSASignContext	X	X
		SECStatus	X	-
	DSA_DestroySignContext()	DSASignContext	-	X
	DSA_CreateVerifyContext()	SECKEYLowPublicKey	X	-
		DSAVerifyContext	-	X
	DSA_Verify()	DSAVerifyContext	X	X
		SECStatus	X	-
	DSA_DestroyVerifyContext()	DSAVerifyContext	-	X
	RSA_Sign()	SECKEYLowPrivateKey	X	-
		SECStatus	X	-
	RSA_CheckSign()	SECKEYLowPublicKey	X	-
		SECStatus	X	-
	RSA_CheckSignRecover()	SECKEYLowPublicKey	X	-
		SECStatus	X	-

	RSA_EncryptBlock()	SECKEYLowPublicKey	X	-
		SECStatus	X	-
	RSA_DecryptBlock()	SECKEYLowPrivateKey	X	-
		SECStatus	X	-
	RSA_SignRaw()	SECKEYLowPrivateKey	X	-
		SECStatus	X	-
	RSA_CheckSignRaw()	SECKEYLowPublicKey	X	-
		SECStatus	X	-
	RSA_CheckSignRecoverRaw()	SECKEYLowPublicKey	X	-
		SECStatus	X	-
	RSA_EncryptBlockRaw()	SECKEYLowPublicKey	X	-
		SECStatus	X	-
	RSA_DecryptBlockRaw()	SECKEYLowPrivateKey	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	DSA_CreateKeyGenContext()	DSAKeyGenContext	-	X
	DSA_KeyGen()	DSAKeyGenContext	X	X
		SECKEYLowPublicKey	-	X
		SECKEYLowPrivateKey	-	X
		SECStatus	X	-
	DSA_DestroyKeyGenContext()	DSAKeyGenContext	-	X

	RSA_NewKey()	RNGContext	X	X
		SECItem	X	X
		SECKEYLowPrivateKey	-	X
PKCS #5 Password-Based Encryption	SEC_PKCS5GetSalt()	SECAlgorithmID	X	X
		SECItem	X	-
	SEC_PKCS5CipherData()	SECAlgorithmID	X	X
		SECItem	X	X
		SECItem	X	-
	SEC_PKCS5CreateAlgorithmID()	SECOidTag	X	-
		SECItem	X	X
		SECAlgorithmID	-	X
PKCS #12 Personal Information Exchange	SEC_PKCS12GetPFX()	SECOidTag	X	-
		SECItem	X	-
	SEC_PKCS12PutPFX()	SECItem	X	-
		SECOidTag	X	-
		SECStatus	X	-
Prime Number Generation	prm_PrimeFind()	CMPInt	X	X
		SECStatus	X	-
	prm_GeneratePrimeRoster()	SECStatus	X	-
	prm_PseudoPrime()	CMPInt	X	-
		SECStatus	-	X
		SECStatus	X	-
	prm_RabinTest()	CMPInt	X	-
		SECStatus	-	X
		SECStatus	X	-
Private Key Storage and Retrieval	SECKEY_CloseKeyDB()	SECKEYKeyDBHandle	X	X
	SECKEY_DeleteKey()	SECKEYKeyDBHandle	X	X
		SECStatus	X	-
	SECKEY_Find()	SECKEYKeyDBHandle	X	X
		SECItem	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()	RNGContext	-	X
		SECStatus	X	-
	RNG_GenerateGlobalRandomBytes()	RNGContext	X	X
		SECStatus	X	-
	RNG_RandomUpdate()	RNGContext	X	X
		SECStatus	X	-
	RNG_ResetRandom()	RNGContext	X	X
		SECStatus	X	-
	RNG_CreateContext()	RNGContext	X	X
		RNGContext	-	X
	RNG_Init()	RNGContext	-	X
	RNG_GenerateRandomBytes()	RNGContext	X	X
		SECStatus	X	-
	RNG_Update()	RNGContext	X	X
		SECStatus	X	-
	RNG_Reseed()	RNGContext	X	X
		SECStatus	X	-
	RNG_DestroyContext()	RNGContext	-	X
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

	SECKEY_UpdateKeyDBPass1()	SECKEYKeyDBHandle	X	X
		SECStatus	X	-
	SECKEY_UpdateKeyDBPass2()	SECKEYKeyDBHandle	X	X
		SECItem	X	X
		SECStatus	X	-
Pseudorandom Number Generation	RNG_RNGInit()	RNGContext	-	X
		SECStatus	X	-
	RNG_GenerateGlobalRandomBytes()	RNGContext	X	X
		SECStatus	X	-
	RNG_RandomUpdate()	RNGContext	X	X
		SECStatus	X	-
	RNG_ResetRandom()	RNGContext	X	X
		SECStatus	X	-
	RNG_CreateContext()	RNGContext	X	X
		RNGContext	-	X
	RNG_Init()	RNGContext	-	X
	RNG_GenerateRandomBytes()	RNGContext	X	X
		SECStatus	X	-
	RNG_Update()	RNGContext	X	X
		SECStatus	X	-
	RNG_Reseed()	RNGContext	X	X
		SECStatus	X	-
	RNG_DestroyContext()	RNGContext	-	X
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. *Netscape 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. A complete reference to such zeroizations is listed in section 8.0 of this document.

1.10 Role-based Authentication

Since all Netscape products utilize role-based authentication, and all products use a single-role mechanism referred to above as a Netscape User, authentication shall always be required upon initializing the FIPS Cryptographic Module. This is true of all Netscape 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 Netscape products since it is only applicable to products attempting to be certified to security level three or four.

Physical Security Considerations

August 25, 1997

Purpose:

To advise organizations of their role in configuring their systems, policies and practices to be consistent with FIPS 140-1 module security, and to describe one method for a user organization to attain level 2 physical security on their general purpose computer systems.

FIPS 140-1 Level 2 Physical Security Requirements

"A cryptographic module shall be designed to employ physical security mechanisms in order to restrict unauthorized physical access to the contents of the module and to deter unauthorized use or unauthorized modification of the module (including substitution of the entire module) when installed. The entire contents of a cryptographic module, including all hardware, firmware, software and data (including plaintext cryptographic keys and unprotected critical security parameters) shall be protected."……

"If the enclosure includes any doors or removable covers, then either they shall be locked with pick-resistant mechanical locks that employ physical or logical keys, or they shall be protected via tamper evident seals……"

User Responsibilities:

It is the responsibility of the user of the cryptographic module to implement the physical protections necessary to provide physical security in accordance with the using organization security policy. This may involve the procurement, application and periodic inspection of tamper evident mechanisms, such as tamper evident seals. In conjunction with the physical protections, the user organization may be obligated to set up policies or procedures to periodically inspect the hardware platforms for evidence of tampering, as well as train operators in what to look for on their systems to identify tamper evidence.

Method:

The FIPS 140-1 Standard allows for the use of seals as a means to provide evidence of a successful or attempted physical attack on a cryptographic module. A recommended method to accomplish this is to appropriately apply holographic seals that are designed specifically for this purpose. The objective of the seals is to provide a high probability of leaving evidence that the enclosure has been breached.

Holographic Seals:

If the use of seals is selected for securing the general purpose computers then a custom or unique holographic image on the seal should be used. The holographic nature of the seals makes them difficult and expensive to copy. The seals should have an aggressive adhesive that is resistant to chemical solvents and should be fragile enough to de-laminate or tear if an attempt is made to lift or remove the seal. The process of removing the seal, should force clear indications of evidence on both the applied surface and the removed seal.

Seal Selection:
When selecting the appropriate seal to optimize tamper evidence, characteristics such as background color, tamper evidence indicators, size and shape are factors. The color of the seal should be different than the color of the product surface. The size and shape should be large enough to have sufficient area for good adhesion.

Location and Positioning:
Determining where the seals will be located and how they will be positioned is important to fulfilling the overall objective of tamper evidence. The primary PC component that should be secured is the enclosure(s) that contains the processor and primary application memory storage. Specifically seals need to be applied to the appropriate interfaces, junctions, and/or fasteners of all doors and covers incorporated in the enclosure design. The seals should be applied to minimize the possibility that the door or cover could be partially opened.

Application Process:
The specific application, materials and process of the seal manufacturer that was selected needs to be strictly adhered to. In general, the application of the seals is a short process.

- Determine appropriate locations on the PC that seals are to be applied
- Clean surfaces
- Apply seals
- Allow adhesive to cure

Cleanliness of surfaces to be bonded are very important for proper adhesion to occur. Often equipment surfaces are contaminated by manufacturing, shipping or office cleaning chemicals resulting in drastically reduced adhesive properties. All surfaces must be appropriately cleaned and dried before the application of the seal. During application, proper pressure needs to be applied to the seal and then allowed to cure before the adhesive reaches specified strength.

Inspection:
Tamper evidence is effective only when the evidence is actually observed by a person that will take action appropriate with the evidence. This means that the user organization needs to implement procedures to train operators on what evidence to look for and what action to take. Periodic inspections or audits on the equipment may also need to be performed by crypto officers. These inspections should all be conducted in accordance with the user organization security policy.