NSS Cryptographic Module

Version 3.11.4

FIPS 140-2 Non-Proprietary Security Policy

Level 1 and 2 Validation

Red Hat, Inc.

Sun Microsystems, Inc.

Document Version 1.19

July 18, 2007
# Table of Contents

Introduction .............................................................................................................................................. 3  
Platform List ........................................................................................................................................... 3  
Note on Calling the API Functions ........................................................................................................ 4  
Security Rules ......................................................................................................................................... 5  
Authentication Policy ............................................................................................................................. 13  
   Specification of Roles ............................................................................................................................. 13  
   Role-Based Authentication ...................................................................................................................... 13  
   Strength of Authentication Mechanism ................................................................................................. 14  
   Multiple Concurrent Operators ............................................................................................................ 14  
Access Control Policy ............................................................................................................................. 15  
   Security-Relevant Information ............................................................................................................... 15  
      Non-NIST-Recommended Elliptic Curves .......................................................................................... 16  
   Specification of Services ....................................................................................................................... 17  
Mitigation of Other Attacks ..................................................................................................................... 25  
Access to Audit Data ............................................................................................................................... 26  
   Access to syslog Log Files .................................................................................................................... 27  
   Access to System Audit Log .................................................................................................................. 27  
   Configure the Solaris Auditing ............................................................................................................... 28  
   Viewing the Audit Trail ......................................................................................................................... 28  
Sample Cryptographic Module Initialization Code .................................................................................. 28  
Acknowledgments ................................................................................................................................... 31  
References ............................................................................................................................................... 31


**Introduction**

The NSS cryptographic module is an open-source, general-purpose cryptographic library, with an API based on the industry standard PKCS #11 version 2.20 [1]. It is available for free under the Mozilla Public License, the GNU General Public License, and the GNU Lesser General Public License. The NSS cryptographic module is jointly developed by Red Hat and Sun engineers and is used in Mozilla Firefox, Thunderbird, and many server applications from Red Hat and Sun.

The NSS cryptographic module has two modes of operation: the *FIPS Approved* mode and *non-FIPS Approved* mode. By default, the module operates in the non-FIPS Approved mode. To operate the module in the FIPS Approved mode, an application must adhere to the security rules in the Security Rules section and initialize the module properly. If an application initializes the NSS cryptographic module by calling the standard PKCS #11 function `C_GetFunctionList` and calls the API functions via the function pointers in that list, it selects the non-FIPS Approved mode. To operate the NSS cryptographic module in the FIPS Approved mode, an application must call the API functions via an alternative set of function pointers. Rule 7 of the Security Rules section specifies how to do this.

In addition, for Security Level 1, the operating system must be configured in a single operator mode of operation by removing all other user accounts and turning off all remote login and access services. If the module is operating at Security Level 2, the environment variable `NSS_ENABLE_AUDIT` must be set to 1 before the application starts.

**Platform List**

FIPS 140-2 conformance testing of the NSS cryptographic module was performed on the seven platforms listed below. The list also specifies the level of elliptic curve cryptography (ECC) support tested on each platform. (The module is configured at compile time with one of the two levels of ECC support: Basic ECC only supports the NIST-recommended curves P-256, P-384, and P-521, whereas Extended ECC supports all the NIST-recommended curves and has additional performance optimizations.)

- Security Level 1
  - Compaq Evo with Pentium 4 CPU (x86), Red Hat Enterprise Linux 4, Basic ECC.
  - Dell Dimension 2400 with Pentium 4 CPU, Microsoft Windows XP SP 2, Basic ECC.
  - Sun W2100z workstation with dual AMD Opteron CPUs, 64-bit Solaris 10, Extended ECC.
  - HP Visualize J5000 Workstation with PA-RISC 2.0 CPU, HP-UX B.11.11 with the HP-UX Strong Random Number Generator (`KRNG11`) bundle installed, Extended ECC.
  - Mac mini with PowerPC G4 CPU, Mac OS X 10.4, Basic ECC.
• Security Level 2
  • IBM xSeries 336 with Intel Xeon CPU (x86_64), Red Hat Enterprise Linux Version 4 Update 1 AS, Extended ECC.

  • Sun Blade 2500 Workstation with UltraSPARC IIIi CPU, Sun Trusted Solaris Version 8 4/01, Extended ECC.

The NSS cryptographic module supports many other platforms. If you would like to have the module validated on other platforms, please contact us.

**Note on Calling the API Functions**

The NSS cryptographic module has two parallel sets of API functions, **FC_***xxx*** and **NSC_***xxx***, that implement the FIPS Approved and non-FIPS Approved modes of operation, respectively. For example, **FC_Initialize** initializes the module's library for the FIPS Approved mode of operation, whereas its counterpart **NSC_Initialize** initializes the library for the non-FIPS Approved mode of operation. All the API functions for the FIPS Approved mode of operation are listed in the **Specification of Services** section.

Among the module's API functions, only **FC_GetFunctionList** and **NSC_GetFunctionList** are exported and therefore callable by their names. (The **C_GetFunctionList** function mentioned in the **Introduction** section is also exported and is just a synonym of **NSC_GetFunctionList**.) All the other API functions must be called via the function pointers returned by **FC_GetFunctionList** or **NSC_GetFunctionList**. **FC_GetFunctionList** and **NSC_GetFunctionList** each return a **CK_FUNCTION_LIST** structure containing function pointers named **C_***xxx*** such as **C_Initialize** and **C_Finalize**. The **C_***xxx*** function pointers in the **CK_FUNCTION_LIST** structure returned by **FC_GetFunctionList** point to the **FC_***xxx*** functions, whereas the **C_***xxx*** function pointers in the **CK_FUNCTION_LIST** structure returned by **NSC_GetFunctionList** point to the **NSC_***xxx*** functions.

For brevity, we use the following convention to describe API function calls. Again we use **FC_Initialize** and **NSC_Initialize** as examples:

  - When we say “call **FC_Initialize**,” we mean “call the FC_Initialize
function via the C_Initialize function pointer in the CK_FUNCTION_LIST structure returned by FC_GetFunctionList.”

- When we say “call NSC_Initialize,” we mean “call the NSC_Initialize function via the C_Initialize function pointer in the CK_FUNCTION_LIST structure returned by NSC_GetFunctionList.”

**Security Rules**

The following list specifies the security rules that the NSS cryptographic module and each product using the module shall adhere to:

1. The NSS cryptographic module shall consist of software libraries compiled for each supported platform.

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 support the NSS User role and the Crypto Officer role.

4. A cryptographic module user shall have access to **all** the services provided by the cryptographic module.

5. Cryptographic module services shall consist of **public services**, which require no user authentication, and **private services**, which require user authentication. Public services do not require access to the secret and private keys and other critical security parameters (CSPs) associated with the user. **Note:** CSPs are security-related information (e.g., secret and private keys, and authentication data such as passwords) whose disclosure or modification can compromise the security of a cryptographic module. Message digesting services are public only when CSPs are not accessed. Services which access CSPs (e.g., FC_GenerateKey, FC_GenerateKeyPair) require authentication.

6. Public key certificates shall be stored in plaintext form because of their public nature.

7. Applications running in the FIPS Approved mode shall call FC_GetFunctionList for the list of function pointers and shall call the API functions via the function pointers in that list for all cryptographic operations. (See the **Note on Calling the API functions** section.) The module changes from FIPS Approved mode to non-FIPS Approved mode when a FC_Finalize/NSC_Initialize sequence is executed; it changes from non-FIPS Approved mode to FIPS Approved mode when a NSC_Finalize/FC_Initialize sequence is executed.

8. In the FIPS Approved mode of operation, the cryptographic module shall enforce
rules specific to FIPS 140-2 requirements.

9. The cryptographic module shall not allow critical errors to compromise security. Whenever a critical error (e.g., a self-test failure) is encountered, the cryptographic module shall enter an error state and the library shall need to be reinitialized to resume normal operation. Reinitialization is accomplished by calling FC_Finalize followed by FC_Initialize.

10. Upon initialization of the cryptographic module library for the FIPS Approved mode of operation, the following power-up self-tests shall be performed:

   a) Triple DES-ECB encrypt/decrypt,
   b) Triple DES-CBC encrypt/decrypt,
   c) AES-ECB encrypt/decrypt (128-bit, 192-bit, and 256-bit keys),
   d) AES-CBC encrypt/decrypt (128-bit, 192-bit, and 256-bit keys),
   e) SHA-1 hash,
   f) SHA-256 hash,
   g) SHA-384 hash,
   h) SHA-512 hash,
   i) HMAC-SHA-1/-SHA-256/-SHA-384/-SHA-512 keyed hash (296-bit key),
   j) RSA encrypt/decrypt (1024-bit modulus $n$),
   k) RSA-SHA-1/-SHA-256/-SHA-384/-SHA-512 signature generation (1024-bit modulus $n$),
   l) RSA-SHA-1/-SHA-256/-SHA-384/-SHA-512 signature verification (1024-bit modulus $n$),
   m) DSA key pair generation (1024-bit prime modulus $p$),
   n) DSA signature generation (1024-bit prime modulus $p$),
   o) DSA signature verification (1024-bit prime modulus $p$),
   p) ECDSA signature generation (Curve P-256; the Extended ECC version of the module also tests Curve K-283),
   q) ECDSA signature verification (Curve P-256; the Extended ECC version of the module also tests Curve K-283),
   r) random number generation, and
   s) software/firmware integrity test (the authentication technique is DSA with 1024-bit prime modulus $p$).

11. Shutting down and restarting the NSS cryptographic module with the FC_Finalize and FC_Initialize functions shall execute the same power-up self-tests detailed above when initializing the module library for the FIPS Approved mode. This allows a user to execute these power-up self-tests on demand as defined in Section 4.9.1 of FIPS 140-2.

12. The NSS cryptographic module shall require the user to establish a password (for the NSS User and Crypto Officer roles) with the FC_InitPIN function in order for subsequent authentications to be enforced. See the Sample Cryptographic Module Initialization Code section below for the sample code to establish the initial user password.

13. A known password check string, encrypted with a Triple-DES key derived from the
password, shall be stored in the private key database (key3.db) in secondary storage. **Note:** this database lies outside the cryptographic boundary. See #16 below.

14. Once a password has been established for the NSS cryptographic module, the module shall allow the user to use the private services if and only if the user successfully authenticates to the module. Password establishment and authentication are required for the operation of the module at both Levels 1 and 2 even though level 1 does not require such authentication method. Password authentication in the Level 1 module does not imply that any of the roles are considered to be authorized for the purposes of Level 2 FIPS 140-2 validation.

15. In order to authenticate to the cryptographic module, the user shall enter the password, and the cryptographic module shall verify that the password is correct 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.

16. The user's password shall act as the key material to encrypt/decrypt secret and private keys. **Note:** Since password-based encryption such as PKCS #5 is not FIPS Approved, password-encrypted secret and private keys should be considered to be in plaintext form in the FIPS Approved mode.

17. Secret and private keys, plaintext passwords, and other security-relevant data items shall be maintained under the control of the cryptographic module. Secret and private keys shall only be passed to higher-level callers in encrypted (wrapped) form with FC_WrapKey using Triple DES or AES (symmetric key algorithms) or RSA (asymmetric key algorithm). **Note:** If the secret and private keys passed to higher-level callers are encrypted using a symmetric key algorithm, the encryption key may be derived from a password. In such a case, they should be considered to be in plaintext form in the FIPS Approved mode.

18. Secret and private keys shall only be stored in encrypted form (using a Triple-DES key derived from the password) in the private key database (key3.db) in secondary storage. **Note:** password-encrypted secret and private keys in the private key database should be considered to be in plaintext form in the FIPS Approved mode.

19. Once the FIPS Approved mode of operation has been selected, the user shall only use the FIPS 140-2 cipher suite.

20. The FIPS 140-2 cipher suite shall consist solely of

- Triple DES (FIPS 46-3) or AES (FIPS 197) for symmetric key encryption and decryption.
- Secure Hash Standard (SHA-1, SHA-256, SHA-384, and SHA-512) (FIPS 180-2) for hashing.
- HMAC (FIPS 198) for keyed hash.
- random number generator (FIPS 186-2 with Change Notice 1).
- Diffie-Hellman, EC Diffie-Hellman, or Key Wrapping using RSA keys for key establishment.
- DSA (FIPS 186-2 with Change Notice 1), RSA (PKCS #1 v2.1), or ECDSA (ANSI X9.62) for signature generation and verification.

**Algorithm validation certificates:**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cert#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triple DES</td>
<td>410 (x86 CPUs)</td>
<td>TECB(e/d; KO 1,2,3); TCBC(e/d; KO 1,2,3)</td>
</tr>
<tr>
<td></td>
<td>469 (non-x86 CPUs)</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>352</td>
<td>ECB(e/d; 128,192,256); CBC(e/d; 128,192,256)</td>
</tr>
<tr>
<td>SHS</td>
<td>426</td>
<td>SHA-1 (BYTE-only) SHA-256 (BYTE-only) SHA-384 (BYTE-only) SHA-512 (BYTE-only)</td>
</tr>
<tr>
<td>HMAC</td>
<td>152</td>
<td>HMAC-SHA1 (Key Sizes Ranges Tested: KS&lt;BS KS=BS KS&gt;BS ) HMAC-SHA256 ( Key Size Ranges Tested: KS&lt;BS KS=BS KS&gt;BS ) HMAC-SHA348 ( Key Size Ranges Tested: KS&lt;BS KS=BS KS&gt;BS ) HMAC-SHA512 ( Key Size Ranges Tested: KS&lt;BS KS=BS KS&gt;BS )</td>
</tr>
<tr>
<td>RNG</td>
<td>208</td>
<td>FIPS 186-2 [ (x-Change Notice); (SHA-1) ] FIPS 186-2 General Purpose [ (x-Change Notice); (SHA-1)) ]</td>
</tr>
<tr>
<td>RSA</td>
<td>152</td>
<td>ALG[RSASSA-PKCS1_V1_5]; SIG(gen); SIG(ver); 1024 , 1536 , 2048 , 3072 , 4096 , SHS: SHA-1 , SHA-256 , SHA-384 , SHA-512</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Cert#</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DSA</td>
<td>172</td>
<td>PQG(gen) MOD( ALL ); PQG(ver) MOD( ALL ); KEYGEN(Y) MOD( ALL ); SIG(gen) MOD( ALL , 960 ); SIG(ver) MOD( ALL );</td>
</tr>
<tr>
<td>ECDSA (Extended ECC)</td>
<td>30</td>
<td>PKG: CURVES( ALL-P ALL-K ALL-B ) PKV: CURVES( ALL-P ALL-K ALL-B ) SIG(gen): CURVES( ALL-P ALL-K ALL-B ) SIG(ver): CURVES( ALL-P ALL-K ALL-B )</td>
</tr>
<tr>
<td>ECDSA (Basic ECC)</td>
<td>37</td>
<td>PKG: CURVES( ALL-P P-256 P-384 P-521 ) PKV: CURVES( ALL-P P-256 P-384 P-521 ) SIG(gen): CURVES( ALL-P P-256 P-384 P-521 ) SIG(ver): CURVES( P-256 P-384 P-521 )</td>
</tr>
</tbody>
</table>

**Caveats:**

- Diffie-Hellman (key agreement, key establishment methodology provides between 80 bits and 112 bits of encryption strength)
- EC Diffie-Hellman (key agreement, key establishment methodology provides between 80 bits and 256 bits of encryption strength)
- RSA (PKCS #1, key wrapping, key establishment methodology provides between 80 bits and 192 bits of encryption strength)

The NSS cryptographic module implements the following non-Approved algorithms, which shall not be used in the FIPS Approved mode of operation:

- RC2 , RC4, or DES for symmetric key encryption and decryption.
- MD2 or MD5 for hashing.

21. Once the FIPS Approved mode of operation has been selected, Triple DES and
AES shall be limited in their use to performing encryption and decryption using either ECB or CBC mode.

22. Once the FIPS Approved mode of operation has been selected, SHA-1, SHA-256, SHA-386, and SHA-512 shall be the only algorithms used to perform one-way hashes of data.

23. Once the FIPS Approved mode of operation has been selected, RSA shall be limited in its use to generating and verifying PKCS #1 signatures, and to encrypting and decrypting key material for key exchange.

24. Once the FIPS Approved mode of operation has been selected, DSA and ECDSA shall be used in addition to RSA to generate and verify signatures.

25. In the FIPS Approved mode of operation, the cryptographic module shall perform a pair-wise consistency test upon each invocation of RSA, DSA, and ECDSA key pair generation as defined in Section 4.9.2 of FIPS 140-2.

26. The cryptographic module shall generate the primes $p$ and $q$ used in the DSA and perform primality test using the algorithms described in Appendix 2 of FIPS 186-2.

27. The cryptographic module shall perform pseudorandom number generation using Algorithm 1 of FIPS 186-2 Change Notice 1, with the one-way function $G$ constructed using SHA-1 and $b$ equal to 256 bits.

28. The cryptographic module shall initialize its pseudorandom number generator by obtaining 1024 bytes of random data from the operating system. The data obtained shall contain at least 256 bits of entropy. Extra entropy input is added by invoking a noise generator. Both initialization and noise generation are specific to the platform on which it was implemented (e.g., Macintosh, UNIX, or Windows). The pseudorandom number generator shall be seeded with noise derived from the execution environment such that the noise is not predictable. The source of noise is considered to be outside the logical boundary of the cryptographic module.

29. A product using the cryptographic module should periodically reseed the module's pseudorandom number generator with unpredictable noise by calling FC_SeedRandom.

30. In the FIPS Approved mode of operation, the cryptographic module shall perform a continuous random number generator test upon each invocation of the pseudorandom number generator as defined in Section 4.9.2 of FIPS 140-2.

31. At level 2 in the FIPS Approved mode of operation the operator shall authenticate successfully before utilizing random number generation services provided by the module. Using random number generation services without authentication will automatically transition the module to the non-Approved mode of operation. The module shall not share CSPs between an Approved mode of operation and a non-Approved mode of operation.

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

33. The cryptographic module shall perform explicit zeroization steps to clear the memory region previously occupied by a plaintext secret key, private key, or password. A plaintext secret or private key shall be zeroized when it is passed to a FC_DestroyObject call. All plaintext secret and private keys shall be zeroized when the NSS cryptographic module is shut down (with a FC_Finalize call) or reinitialized (with a FC_InitToken call), or when the state changes between the FIPS Approved mode and non-FIPS Approved mode (with a NSC_Finalize/FC_Initialize or FC_Finalize/NSC_Initialize sequence).

All zeroization shall be performed by storing the value 0 into every byte of the memory region previously occupied by a plaintext secret key, private key, or password.

34. The NSS cryptographic module consists of the following shared libraries/DLLs and the associated .chk files:

- Windows XP Service Pack 2
  - softokn3.dll
  - softokn3.chk
  - freebl3.dll
  - freebl3.chk
- 32-bit HP-UX B.11.11 PA-RISC 2.0
  - libsoftokn3.sl
  - libsoftokn3.chk
  - libfreebl_32fpu_3.sl
  - libfreebl_32fpu_3.chk
- Mac OS X 10.4
  - libsoftokn3.dylib
  - libsoftokn3.chk
  - libfreebl3.dylib
  - libfreebl3.chk
- 64-bit Trusted Solaris 8 UltraSPARC IIIi
  - libsoftokn3.so
  - libsoftokn3.chk
  - libfreebl_64fpu_3.so
  - libfreebl_64fpu_3.chk
- 64-bit Solaris 10 AMD64, Red Hat Enterprise Linux 4 x86, and Red Hat Enterprise Linux 4 x86_64
  - libsoftokn3.so
  - libsoftokn3.chk
  - libfreebl3.so
  - libfreebl3.chk

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 is outside the cryptographic boundary because none of the NSPR functions are security-relevant. NSPR consists of the following shared libraries/DLLs:

- Windows XP Service Pack 2
  - plc4.dll
  - plds4.dll
  - nspr4.dll
- HP-UX B.11.11 PA-RISC 2.0
  - libplc4.sl
  - libplds4.sl
  - libnspr4.sl
- Mac OS X 10.4
  - libplc4.dylib
  - libplds4.dylib
  - libnspr4.dylib
- 64-bit Solaris 10 AMD64, 64-bit Trusted Solaris 8 UltraSPARC IIIi, Red Hat Enterprise Linux 4 x86, and Red Hat Enterprise Linux 4 x86_64
  - libplc4.so
  - libplds4.so
  - libnspr4.so

The installation instructions are:

**Step 1:** Install the shared libraries/DLLs and the associated .chk files in a directory on the shared library/DLL search path, which could be a system library directory (/usr/lib on Unix/Linux or C:\WINDOWS\system32 on Windows) or a directory specified in the following environment variable:

- Windows XP Service Pack 2: PATH
- HP-UX B.11.11: SHLIB_PATH
- Mac OS X 10.4: DYLD_LIBRARY_PATH
- Solaris and Linux: LD_LIBRARY_PATH

**Step 2:** Use the chmod utility to set the file mode bits of the shared libraries/DLLs to 0755 so that all users can execute the library files, but only the files' owner can modify (i.e., write, replace, and delete) the files. For example, on most Unix and Linux platforms,

$ chmod 0755 libsoftokn3.so libfreebl*3.so libplc4.so libplds4.so libnspr4.so

The discretionary access control protects the binaries stored on disk from being tampered with.
Step 3: Use the `chmod` utility to set the file mode bits of the associated .chk files to 0644. For example, on most Unix and Linux platforms,

```
$ chmod 0644 libsoftkn3.chk libfreebl*3.chk
```

Step 4: As specified in Rule 7, to operate the NSS cryptographic module in the FIPS Approved mode, an application must call the alternative PKCS #11 function `FC_GetFunctionList` and call the API functions via the function pointers in that list. The user shall initialize the password when using the module for the first time. Before the user password is initialized, access to the module shall be controlled. See the Sample Cryptographic Module Initialization Code section below for sample code.

(End of Security Rules)

**Authentication Policy**

**Specification of Roles**

The NSS cryptographic module supports two authorized roles for operators.

- The NSS User role provides access to all cryptographic and general-purpose services (except those that perform an initialization function) and all keys stored in the private key database. An NSS User utilizes secure services and is also responsible for the retrieval, updating, and deletion of keys from the private key database.

- The Crypto Officer role is supported for the installation and initialization of the module. The Crypto Officer must control the access to the module both before and after installation. Control consists of management of physical access to the computer executing the NSS cryptographic module code as well as management of the security facilities provided by the operating system.

The NSS cryptographic module uses a combined role approach -- by authenticating to the module, an operator assumes both the NSS User role and the Crypto Officer role at the same time.

The NSS cryptographic module does not have a maintenance role.

**Role-Based Authentication**

The NSS cryptographic module uses role-based authentication to control access to the module. To perform sensitive services using the cryptographic module, an operator must
explicitly request to assume the NSS User role by logging into the module and performing an authentication procedure using information unique to that operator (password). The password is initialized by the Crypto Officer as part of module initialization. Role-based authentication is used to safeguard a user's private key information. However, discretionary access control is used to safeguard all other information (e.g., the public key certificate database).

Authentication shall always be required upon initializing the NSS cryptographic module library in the FIPS Approved mode. If a function that requires authentication is called before the operator is authenticated, it returns the CKR_USER_NOT_LOGGED_IN error code. Call the FC_Login function to provide the required authentication. The only exception to this is the random number generator function. The Level 2 module cannot be used in FIPS mode if the NSS User role is not authenticated and the random number generator is called from this role.

**Strength of Authentication Mechanism**

In the FIPS Approved mode, the NSS cryptographic module imposes the following requirements on the password. These requirements are enforced by the module on password initialization or change.

- The password must be at least seven characters long.
- The password must consist of characters from three or more character classes. We define five character classes: digits (0-9), ASCII lowercase letters, ASCII uppercase letters, ASCII non-alphanumeric characters (such as space and punctuation marks), and non-ASCII characters. If an ASCII uppercase letter is the first character of the password, the uppercase letter is not counted toward its character class. Similarly, if a digit is the last character of the password, the digit is not counted toward its character class.

To estimate the probability that a random guess of the password will succeed, we assume that

- the characters of the password are independent with each other, and
- the probability of guessing an individual character of the password is less than 1/10.

Since the password is at least 7 characters long, the probability that a random guess of the password will succeed is less than \((1/10)^7 = 1/10,000,000\).

After each failed authentication attempt in the FIPS Approved mode, the NSS cryptographic module inserts a one-second delay before returning to the caller, allowing at most 60 authentication attempts during a one-minute period. Therefore, the probability of a successful random guess of the password during a one-minute period is less than \(60 \times 1/10,000,000 = 0.6 \times (1/100,000)\).
Multiple Concurrent Operators

The NSS cryptographic module doesn't allow concurrent operators.

- For Security Level 1, the operating system has been restricted to a single operator mode of operation, so concurrent operators are explicitly excluded (FIPS 140-2 Section 4.6.1).
- On a multi-user operating system, this is enforced by making the NSS certificate and private key databases readable and writable by the owner of the files only.

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

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

Access Control Policy

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

Security-Relevant Information

The NSS cryptographic module employs the following cryptographic keys and CSPs in the FIPS Approved mode of operation. Note that the private key database (key3.db) mentioned below is outside the cryptographic boundary.

- AES secret keys: The module supports 128-bit, 192-bit, and 256-bit AES keys. The keys may be stored in memory or in the private key database (key3.db).
- Triple-DES secret keys: 168-bit. The keys may be stored in memory or in the private key database (key3.db).
- HMAC secret keys: HMAC key size must be greater than or equal to half the size of the hash function output. The keys may be stored in memory or in the private key database (key3.db).
- DSA public keys and private keys: The module supports DSA key sizes of 512-1024 bits. DSA keys of 1024 bits shall be used in the FIPS Approved mode of operation. The keys may be stored in memory or in the private key database (key3.db).
- RSA public keys and private keys (used for digital signatures and key transport): The module supports RSA key sizes of 1024-8192 bits. The keys may be stored in memory or in the private key database (key3.db).
• EC public keys and private keys (used for ECDSA digital signatures and EC Diffie-Hellman key agreement): The module supports elliptic curve key sizes of 256-521 bits in the Basic ECC version and 163-571 bits in the Extended ECC version. EC keys of 160 bits or higher shall be used in the FIPS Approved mode of operation. (See the Non-NIST-Recommended Elliptic Curves section below.) The keys may be stored in memory or in the private key database (key3.db).
• Diffie-Hellman public keys and private keys (used for key agreement): The module supports Diffie-Hellman public key sizes of 1024-2236 bits. The keys may be stored in memory or in the private key database (key3.db).
• TLS premaster secret (used in deriving the TLS master secret): 48-byte. Stored in memory.
• TLS master secret (a secret shared between the peers in TLS connections, used in the generation of symmetric cipher keys, IVs, and MAC secrets for TLS): 48-byte. Stored in memory.
• seed-key of the Approved random number generator: 256-bit. Stored in memory.
• authentication data (passwords): Stored in the private key database (key3.db).
• audited events and audit data (Security Level 2 only): Stored in the system audit logs.

Note: The NSS cryptographic module does not implement the TLS protocol. The NSS cryptographic module implements the cryptographic operations, including TLS-specific key generation and derivation operations, that can be used to implement the TLS protocol.

Non-NIST-Recommended Elliptic Curves

The Basic ECC version of the NSS cryptographic module only implements the NIST-recommended elliptic curves P-256, P-384, and P-521 specified in FIPS 186-2.

The Extended ECC version of the NSS cryptographic module implements all the NIST-recommended elliptic curves and the following non-NIST-recommended curves:

<table>
<thead>
<tr>
<th>Curve family</th>
<th>Curve names</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI X9.62-1998 prime curves</td>
<td>prime192v2, prime192v3, prime239v1, prime239v2, and prime239v3</td>
</tr>
<tr>
<td>ANSI X9.62-1998 binary curves</td>
<td>c2pnb163v1, c2pnb163v2, c2pnb163v3, c2tnb191v1, c2tnb191v2, c2tnb191v3, c2tnb239v1, c2tnb239v2, c2tnb239v3, c2tnb359v1, and c2tnb431r1</td>
</tr>
<tr>
<td>SEC 2 prime curves</td>
<td>secp160k1, secp160r1, secp160r2, secp192k1, secp224k1, and secp256k1</td>
</tr>
<tr>
<td>SEC 2 binary curves</td>
<td>sect163r1, sect193r1, sect193r2, and sect239k1</td>
</tr>
</tbody>
</table>
Although FIPS 140-2 Implementation Guidance Section 1.6 allows the use of non-NIST-recommended curves in the FIPS Approved mode of operation, we recommend that the non-NIST-recommended curves not be used in the FIPS Approved mode.

The Extended ECC version of the NSS cryptographic module also implements the following non-NIST-recommended curves, which shall not be used in the FIPS Approved mode.

<table>
<thead>
<tr>
<th>Curve family</th>
<th>Curve names</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI X9.62-1998 binary curves</td>
<td>c2pnb176w1, c2pnb208w1, c2pnb272w1, c2pnb304w1, and c2pnb368w1</td>
<td>disallowed in ANSI X9.62-2005</td>
</tr>
<tr>
<td>SEC 2 prime curves</td>
<td>secp112r1, secp112r2, secp128r1, and secp128r2</td>
<td>key sizes smaller than 160 bits</td>
</tr>
<tr>
<td>SEC 2 binary curves</td>
<td>sect113r1, sect113r2, sect131r1, and sect131r2</td>
<td>key sizes smaller than 160 bits</td>
</tr>
</tbody>
</table>

**Specification of Services**

Some services require the user to assume the Crypto Officer or NSS User role. In the table below, the role is specified for each service. If the Role column is blank, no role needs to be assumed for that service; such a service (e.g., random number generation and hashing) does not affect the security of the module because it does not require access to the secret and private keys and other CSPs associated with the user. The table lists each service as an API function and correlates role, service type, and type of access to the cryptographic keys and CSPs. Access types **R**, **W**, and **Z** stand for Read, Write, and Zeroize, respectively.
<table>
<thead>
<tr>
<th>Service Category</th>
<th>Role</th>
<th>Function Name</th>
<th>Description</th>
<th>Cryptographic Keys and CSPs Accessed</th>
<th>Access type, RWZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIPS 140-2 specific</td>
<td></td>
<td>FC_GetFunctionList</td>
<td>returns the list of function pointers for the FIPS Approved mode of operation</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>Module Initialization</td>
<td>Crypto Officer</td>
<td>FC_InitToken</td>
<td>initializes or reinitializes a token</td>
<td>password and all keys</td>
<td>Z</td>
</tr>
<tr>
<td></td>
<td>Crypto Officer</td>
<td>FC_InitPIN</td>
<td>initializes the user's password, i.e., sets the user's initial password</td>
<td>password</td>
<td>W</td>
</tr>
<tr>
<td>General purpose</td>
<td></td>
<td>FC_Initialize</td>
<td>initializes the module library for the FIPS Approved mode of operation. This function provides the power-up self-test service.</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_Finalize</td>
<td>finalizes (shuts down) the module library</td>
<td>all keys</td>
<td>Z</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_GetInfo</td>
<td>obtains general information about the module library</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>Service Category</td>
<td>Role</td>
<td>Function Name</td>
<td>Description</td>
<td>Cryptographic Keys and CSPs Accessed</td>
<td>Access type, RWZ</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Slot and token management</td>
<td></td>
<td>FC_GetSlotList</td>
<td>obtains a list of slots in the system</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_GetSlotInfo</td>
<td>obtains information about a particular slot</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_GetTokenInfo</td>
<td>obtains information about the token. This function provides the Show Status service.</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_WaitForSlotEvent</td>
<td>This function is not supported by the NSS cryptographic module.</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_GetMechanismList</td>
<td>obtains a list of mechanisms (cryptographic algorithms) supported by a token</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_GetMechanismInfo</td>
<td>obtains information about a particular mechanism</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>NSS User</td>
<td></td>
<td>FC_SetPIN</td>
<td>changes the user's password</td>
<td>password</td>
<td>RW</td>
</tr>
<tr>
<td>Service Category</td>
<td>Role</td>
<td>Function Name</td>
<td>Description</td>
<td>Cryptographic Keys and CSPs Accessed</td>
<td>Access type, RWZ</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------</td>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Session management</td>
<td></td>
<td>FC_OpenSession</td>
<td>opens a connection (&quot;session&quot;) between an application and a particular token</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_CloseSession</td>
<td>closes a session</td>
<td>keys of the session</td>
<td>Z</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_CloseAllSessions</td>
<td>closes all sessions with a token</td>
<td>all keys</td>
<td>Z</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_GetSessionInfo</td>
<td>obtains information about the session. This function provides the Show Status service.</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_GetOperationState</td>
<td>saves the state of the cryptographic operation in a session. This function is only implemented for message digest operations.</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_SetOperationState</td>
<td>restores the state of the cryptographic operation in a session. This function is only implemented for message digest operations.</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_Login</td>
<td>logs into a token</td>
<td>password</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_Logout</td>
<td>logs out from a token</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>Service Category</td>
<td>Role</td>
<td>Function Name</td>
<td>Description</td>
<td>Cryptographic Keys and CSPs Accessed</td>
<td>Access type, RWZ</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------</td>
<td>-----------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Object management</td>
<td>NSS User</td>
<td>FC_CreateObject</td>
<td>creates an object</td>
<td>key</td>
<td>W</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_CopyObject</td>
<td>creates a copy of an object</td>
<td>original key, new key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_DestroyObject</td>
<td>destroys an object</td>
<td>key</td>
<td>Z</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_GetObjectSize</td>
<td>obtains the size of an object</td>
<td>key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_GetAttributeValue</td>
<td>obtains an attribute value of an object</td>
<td>key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_SetAttributeValue</td>
<td>modifies an attribute value of an object</td>
<td>key</td>
<td>W</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_FindObjectsInit</td>
<td>initializes an object search operation</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_FindObjects</td>
<td>continues an object search operation</td>
<td>keys matching the search criteria</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_FindObjectsFinal</td>
<td>finishes an object search operation</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>Service Category</td>
<td>Role</td>
<td>Function Name</td>
<td>Description</td>
<td>Cryptographic Keys and CSPs Accessed</td>
<td>Access Type, RWZ</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Encryption and decryption</td>
<td>NSS User</td>
<td>FC_EncryptInit</td>
<td>initializes an encryption operation</td>
<td>encryption key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_Encrypt</td>
<td>encrypts single-part data</td>
<td>encryption key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_EncryptUpdate</td>
<td>continues a multiple-part encryption operation</td>
<td>encryption key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_EncryptFinal</td>
<td>finishes a multiple-part encryption operation</td>
<td>encryption key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_DecryptInit</td>
<td>initializes a decryption operation</td>
<td>decryption key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_Decrypt</td>
<td>decrypts single-part encrypted data</td>
<td>decryption key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_DecryptUpdate</td>
<td>continues a multiple-part decryption operation</td>
<td>decryption key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_DecryptFinal</td>
<td>finishes a multiple-part decryption operation</td>
<td>decryption key</td>
<td>R</td>
</tr>
<tr>
<td>Message digesting</td>
<td></td>
<td>FC_DigestInit</td>
<td>initializes a message-digesting operation</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_Digest</td>
<td>digests single-part data</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_DigestUpdate</td>
<td>continues a multiple-part digesting operation</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>NSS User (see the note at the end of the table)</td>
<td>FC_DigestKey</td>
<td>continues a multi-part message-digesting operation by digesting the value of a secret key as part of the data already digested</td>
<td>key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FC_DigestFinal</td>
<td>finishes a multiple-part digesting operation</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>Service Category</td>
<td>Role</td>
<td>Function Name</td>
<td>Description</td>
<td>Cryptographic Keys and CSPs Accessed</td>
<td>Access type, RWZ</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------</td>
<td>----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Signature and verification</td>
<td>NSS User</td>
<td>FC_SignInit</td>
<td>initializes a signature operation</td>
<td>signing/HMAC key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_Sign</td>
<td>signs single-part data</td>
<td>signing/HMAC key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_SignUpdate</td>
<td>continues a multiple-part signature operation</td>
<td>signing/HMAC key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_SignFinal</td>
<td>finishes a multiple-part signature operation</td>
<td>signing/HMAC key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_SignRecoverInit</td>
<td>initializes a signature operation, where the data can be recovered from the signature</td>
<td>RSA signing key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_SignRecover</td>
<td>signs single-part data, where the data can be recovered from the signature</td>
<td>RSA signing key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_VerifyInit</td>
<td>initializes a verification operation</td>
<td>Verification/HMAC key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_Verify</td>
<td>verifies a signature on single-part data</td>
<td>verification/HMAC key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_VerifyUpdate</td>
<td>continues a multiple-part verification operation</td>
<td>verification/HMAC key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_VerifyFinal</td>
<td>finishes a multiple-part verification operation</td>
<td>verification/HMAC key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_VerifyRecoverInit</td>
<td>initializes a verification operation where the data is recovered from the signature</td>
<td>RSA verification key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_VerifyRecover</td>
<td>verifies a signature on single-part data, where the data is recovered from the signature</td>
<td>RSA verification key</td>
<td>R</td>
</tr>
<tr>
<td>Service Category</td>
<td>Role</td>
<td>Function Name</td>
<td>Description</td>
<td>Cryptographic Keys and CSPs Accessed</td>
<td>Access type, RWZ</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-------</td>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Dual-function cryptographic operations</td>
<td>NSS User</td>
<td>FC_DigestEncryptUpdate</td>
<td>continues a multiple-part digesting and encryption operation</td>
<td>encryption key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_DecryptDigestUpdate</td>
<td>continues a multiple-part decryption and digesting operation</td>
<td>decryption key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_SignEncryptUpdate</td>
<td>continues a multiple-part signing and encryption operation</td>
<td>signing/HMAC key, encryption key</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_DecryptVerifyUpdate</td>
<td>continues a multiple-part decryption and verify operation</td>
<td>decryption key, verification/HMAC key</td>
<td>R</td>
</tr>
<tr>
<td>Key management</td>
<td>NSS User</td>
<td>FC_GenerateKey</td>
<td>generates a secret key (used by TLS to generate premaster secrets)</td>
<td>key</td>
<td>W</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_GenerateKeyPair</td>
<td>generates a public/private key pair. This function performs the pair-wise consistency tests.</td>
<td>key pair</td>
<td>W</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_WrapKey</td>
<td>wraps (encrypts) a key</td>
<td>wrapping key, key to be wrapped</td>
<td>R, R</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_UnwrapKey</td>
<td>unwraps (decrypts) a key</td>
<td>unwrapping key, unwrapped key</td>
<td>R, W</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_DeriveKey</td>
<td>derives a key from a base key (used by TLS to derive keys from the master secret)</td>
<td>base key, derived key</td>
<td>R, W</td>
</tr>
<tr>
<td>Service Category</td>
<td>Role</td>
<td>Function Name</td>
<td>Description</td>
<td>Cryptographic Keys and CSPs Accessed</td>
<td>Access type, RWZ</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Random number generation</td>
<td>NSS User</td>
<td>FC_SeedRandom</td>
<td>mixes in additional seed material to the random number generator</td>
<td>RNG seed-key</td>
<td>RW</td>
</tr>
<tr>
<td></td>
<td>NSS User</td>
<td>FC_GenerateRandom</td>
<td>generates random data. This function performs the continuous random number generator test.</td>
<td>RNG seed-key</td>
<td>RW</td>
</tr>
<tr>
<td>Parallel function</td>
<td></td>
<td>FC_GetFunctionStatus</td>
<td>a legacy function, which simply returns the value 0x00000051 (function not parallel)</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>management</td>
<td></td>
<td>FC_CancelFunction</td>
<td>a legacy function, which simply returns the value 0x00000051 (function not parallel)</td>
<td>none</td>
<td>-</td>
</tr>
</tbody>
</table>

**Note:** The message digesting functions (except FC_DigestKey) don't require the user to assume an authorized role because they don't 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.

**Mitigation of Other Attacks**

The NSS cryptographic module is designed to mitigate the following attacks.
### Access to Audit Data

The NSS cryptographic module may use the Unix `syslog` function and the audit mechanism provided by the operating system to audit events. (Auditing is not yet implemented on Windows.) Auditing is turned off by default. To turn on the auditing...
capability, you need to set the environment variable NSS_ENABLE_AUDIT to 1. You also need to configure the operating system's audit mechanism.

Access to the audit data is described in the next two subsections.

**Access to syslog Log Files**

On Unix (including Linux and Mac OS X), the NSS cryptographic module uses the syslog function to audit events, so the audit data are stored in the system log. Only the root user can modify the system log. On some platforms, only the root user can read the system log. On other platforms, all users can read the system log.

The system log is usually under the /var/adm or /var/log directory. The exact location of the system log is specified in the /etc/syslog.conf file. The NSS cryptographic module uses the default user facility and the info, warning, and err severity levels for its log messages. We give two examples below.

**Red Hat Enterprise Linux 4:** The /etc/syslog.conf file on Red Hat Enterprise Linux 4 has:

```plaintext
*.info;mail.none;authpriv.none;cron.none        /var/log/messages
```

which specifies that /var/log/messages is the system log.

**Solaris 10:** The /etc/syslog.conf file on Solaris 10 has:

```plaintext
*.err;kern.debug;daemon.notice;mail.crit        /var/adm/messages
```

which specifies that /var/adm/messages is the system log.

**Access to System Audit Log**

To meet the audit requirements of FIPS 140-2 at Security Level 2, on Red Hat Enterprise Linux 4 and Trusted Solaris 8, the NSS cryptographic module also uses the audit mechanism provided by the operating system to audit events. The audit data are stored in the system audit log. Only the root user can read or modify the system audit log.

On Red Hat Enterprise Linux 4, the system audit log is in the /var/log/audit directory. On Solaris, default audit records are stored in /var/audit/.
Configure the Solaris Auditing

To configure the system audit mechanism on Solaris, the following administration tasks need to be completed. Create the audit class 'fp', then create the audit event 'AUE_FIPS_AUDIT' and add the class 'fp' to the audit_control file.

Edit /etc/security/audit_class add line: 0x99000000:fp:NSS FIPS Security Msgs

Edit /etc/security/audit_event add line: 34444:AUE_FIPS_AUDIT:fp

Edit /etc/security/audit_control add 'fp' to the "flags:" as in: flags:lo,ap,fp

On Trusted Solaris 8, auditing is enabled by default; for non-trusted Solaris run: /etc/security/bsmconv (either as root or a user that has been given the Audit Control RBAC profile in Solaris 8) and reboot your system. After the system has rebooted, ensure auditd is running: ps -ecf | grep auditd

Viewing the Audit Trail

By default, the audit logs are stored in /var/audit. To view the active audit trail, ensure there is only one *not_terminated* audit file. If there are others, delete the older ones before executing this command.

1. cd /var/audit
2. tail -0f *not_terminated* | praudit

Note: On Trusted Solaris 8 you need to assume a role with the tail and praudit commands with the proc_audit_appl and proc_audit_tcb privileges.

You can also view the existing audit files using auditreduce.

1. cd /var/audit
2. auditreduce -m 34444 *not_terminated* | praudit -l

Sample Cryptographic Module Initialization Code

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

```c
#include "prlink.h"
#include "cryptoki.h"
#include <assert.h>
```
```c
#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`.

**Acknowledgments**

Matthew Harmsen, John Hines, Ian McGreer, and Bishakha Banerjee wrote previous versions of this document. Julien Pierre and Steve Parkinson's review comments improved the presentation and accuracy of the information. The current version was written by Wan-Teh Chang, Glen Beasley and Neil Williams.

**References**


