NSS Freebl Cryptographic Module

Version 3.12.9.1

FIPS 140-2 Non-Proprietary Security Policy

Level 1 Validation

Red Hat, Inc.

Document Version 1.3

January25, 2012

Table of Contents

Introduction

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 140-2 standard, and the additional security rules listed below. The rules of operation of the cryptographic module that define within which role(s), and under what circumstances (when performing which services), an operator is allowed to maintain or disclose each security relevant data item of the cryptographic module.

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

- To induce the cryptographic module vendor 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.

The NSS Freebl cryptographic module is an open-source, general-purpose cryptographic hash library. It is available for free under the Mozilla Public License, the GNU General Public License, and the GNU Lesser General Public License. The NSS Freebl cryptographic module is jointly developed by Red Hat and Sun engineers and is used in the GNU glibc library.

The NSS Freebl 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, initialize the module properly with the fips_enabled flag in the Linux kernel set to true.

In addition 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.

This module is a multi-chip standalone module, and no components in the module are excluded from FIPS 140-2 security requirements.

Platform List

FIPS 140-2 conformance testing of the NSS Freebl cryptographic module was performed on the following platforms listed below.

- Security Level 1
  - `64-bit binary on Intel Core i7 running Red Hat Enterprise Linux v6.2`
  - `32-bit binary on Intel Core i7 running Red Hat Enterprise Linux v6.2`

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

Security Rules

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

1. The NSS Freebl cryptographic module shall consist of a software library 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. A cryptographic module user shall have access to **all** the services provided by the cryptographic module.
3. Message digesting services are public only since CSPs are not accessed.
4. In the FIPS Approved mode of operation, the cryptographic module shall enforce rules specific to FIPS 140-2 requirements.
5. In the FIPS Approved mode of operation the cryptographic module shall not allow critical errors to compromise security. Whenever a self-test failure is encountered, the cryptographic module shall enter an error state and the library shall need to be unloaded from memory by terminating the application and then reloaded into memory by restarting the application and then reinitializing the module in FIPS-Approved mode by calling `NSSLOW_Init`.
6. Restarting the NSS Freebl cryptographic module by calling the functions `NSSLOW_Shudown` and `NSSLOW_Init` shall execute the same power-up self-tests as also performed 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.
7. The FIPS 140-2 cipher suite shall consist solely of Secure Hash Standard (SHA-1, SHA-256, SHA-384, and SHA-512) (FIPS 180-2) for hashing.

   Algorithm validation certificates:

Algorithm	Cert#	Description
SHS	1675	SHA-1 (BYTE-only) SHA-256 (BYTE-only) SHA-384 (BYTE-only) SHA-512 (BYTE-only)
DSA	602	DSA 1024 used internally for integrity checking

   The NSS Freebl cryptographic module implements the following non-Approved algorithms, which an operator shall not use while operating the module in a FIPS compliant manner:

   - MD2 or MD5 for hashing.

8. 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.

9. The NSS Freebl cryptographic module consists of the following shared library and the associated `.chk` file:

- Red Hat Enterprise Linux 6.2 x86, and Red Hat Enterprise Linux 6.2 x86_64
    - `libfreebl3.so`
    - `libfreebl3.chk`

`The` installation instructions to be followed by the Crypto-Officer are:

Obtain the validated version of the module (3.12.9.1 either from Red Hat in the form of a pre-compiled executable distribution [rpm] or from Mozilla directly [http://developr.mozilla.org/en/NSS_3.12.9_release_notes] and install the module.

For the operational environment used during validation testing (Red hat Linux 6.2), Red Hat Makes a pre-compiled rpm package available (nss-softoken-freebl-3.12.9.-11.el6.rpm) and one can install this package with the command " rpm -U nss-softoken-freebl-3.12.9-11.el6.rpm" or even "yum update nss-softoken-freebl" if the Red Hat Enterprise Server 6.2's yum repository is connected to the Red Hat Network. After install the library, follow the steps below.

Step 1: Ensure that the shared library and the associated `.chk` are installed file in a directory on the library search path, which could be a system library directory (`/usr/lib` on Unix/Linux) or a directory specified in the following environment variable:

- Linux: `LD_LIBRARY_PATH`

An install using rpm automatically takes care of the proper location of the install files.

Step 2: Use the `chmod` utility to set the file mode bits of the shared library 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 libfreebl3.so
```

The discretionary access control protects the binaries stored on disk from being tampered with.

An install using rpm automatically takes care of the proper permissions of the library.

Step 3: Use the `chmod` utility to set the file mode bits of the associated `.chk` file to **0644**. For example, on most Unix and Linux platforms,

```
$ chmod 0644 libfreebl3.chk
```

An install using rpm automatically takes care of the proper permissions of the library.

Step 4: The kernel fips_enabled flag must be set to '1'. The NSS Freebl cryptographic module detects this by reading /proc/sys/crypto/fips_enabled.

To set the kernel fips_enabled flag, perform the following:
1. Install the dracut-fips package:

```
# yum install dracut-fips
```

2. Recreate the INITRAMFS image:

```
# dracut -f
```

3. After regenerating the initramfs, the crypto officer has to append the following string to the kernel command line by changing the setting in the boot loader, which can be done by editing the `/etc/grub.conf` file on Red Hat Enterprise Linux:

```
fips=1
```

Additionally, if /boot or /boot/efi resides on a separate partition, the kernel parameter `boot=<partition of /boot or /boot/efi>` must also be supplied along with the above mentioned “fips=1” string. The partition can be identified with the command "df /boot" or "df /boot/efi" respectively. For example:

```
$ df /boot
Filesystem      1K-blocks  Used     Available     Use%  Mounted on
/dev/sda1       233191     30454    190296        14%   /boot
```

The partition of /boot is located on /dev/sda1 in this example. Therefore, the kernel command line in the /etc/grub.conf file should be appended with the following string:

```
"fips =1  boot=/dev/sda1"
```

5. Once the bootloader is updated, it is necessary to run dracut -f again.

Reboot to apply these settings.

After following the above steps, the module has been successful initialized in the FIPS-Approved mode and all of the module’s APIs can be accessed by the User role. In order to start using the module in FIPS-mode, an application must call NSSLOW_Init and ensure that it returns a non-NULL poiinter. A NULL pointer returned by this API indicates that the module has failed one or the power-up self-tests and entered the error-state.

(End of Security Rules)

Authentication Policy

Specification of Roles

The NSS Freebl cryptographic module supports two authorized roles for operators.

- The NSS User role provides access to all module services specified in the 'Specfication of Services' table on page 11.

- The Crypto Officer role is supported for installing, uninstalling, and controlling access to 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 Freebl cryptographic module code as well as management of the security facilities provided by the operating system.

The NSS Freebl cryptographic module does not have a maintenance role. The roles are implicitly-assumed depending on the module service being executed, as the module does not implement any operator authentication.

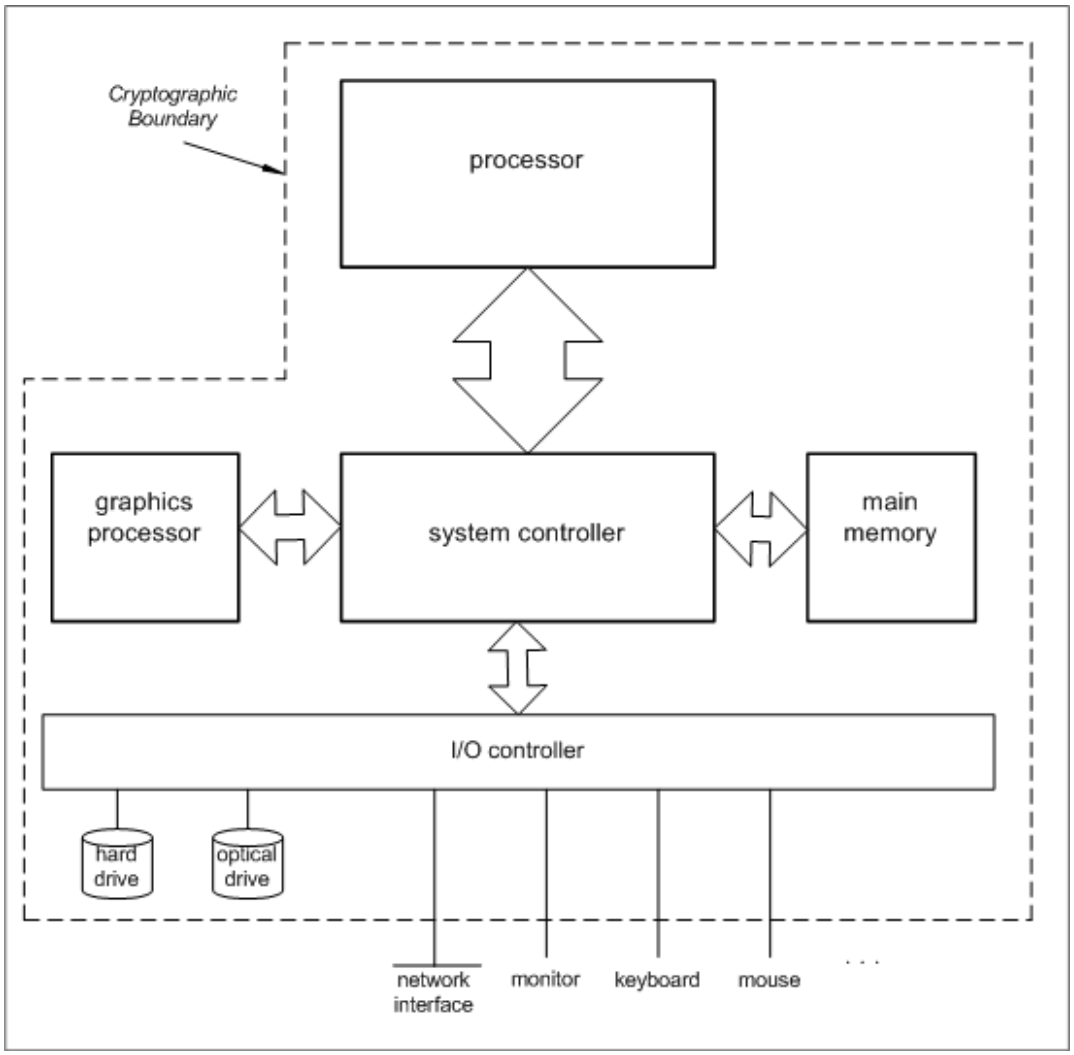
Multiple Concurrent Operators

The NSS Freebl cryptographic module doesn't allow concurrent operators.

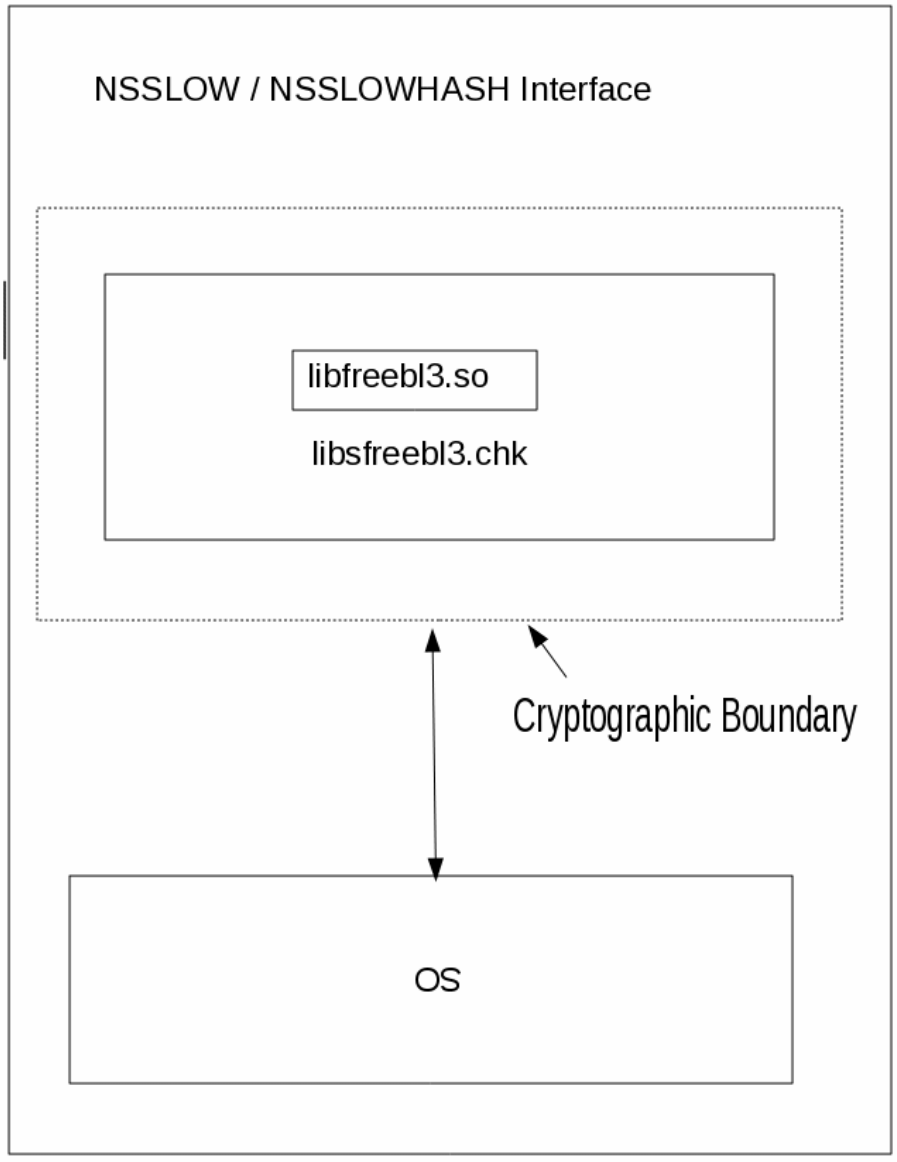
- The operating system has been restricted to a single operator mode of operation, so concurrent operators are explicitly excluded (FIPS 140-2 Section 4.6.1).

Module Ports and Interfaces

Physical Cryptographic Boundary



Logical Cryptographic Boundary



Logical Interfaces

The module's interfaces consist of only the APIs declared in the NSSLOW / NSSLOWHASH interface (i.e. nsslowhash.h). The following four logical interfaces have been designed within the NSS Freebl cryptographic module.

1. Data input interface: function input arguments that specify plaintext data; and hash data that are to be input to and processed by the NSS Freebl cryptographic module.

2. Data output interface: function output arguments that receive plaintext data; and hash data from the NSS Freebl cryptographic module.

3. Control input interface: function calls, or input arguments that specify commands and control data (e.g., algorithms, algorithm modes, or module settings) used to control the operation of the NSS Freebl cryptographic module

4. Status output interface: function return codes, error codes, or output arguments that receive status information used to indicate the status of the NSS Freebl cryptographic module

The NSS Freebl cryptographic module uses different function arguments for input and output to distinguish between data and control for input and data and status for output, and to disconnect the logical paths followed by data/control entering the module and data/status exiting the module. The NSS Freebl cryptographic module doesn't use the same buffer for input and output. After the NSS Freebl cryptographic module is done with an input buffer that holds security-related information, it always zeroizes the buffer so that if the memory is later reused as an output buffer, no sensitive information may be inadvertently leaked.

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. The NSS Freebl cryptographic module contains only one CSP, which is the DSA public key. This is the 1024-bit integrity verification key that the module uses to verify its integrity only during the power-up self-tests. None of the module's services allow access to this CSP to any operator.

Security-Relevant Information

Self-Tests

While in FIPS-Approved mode, the module's initialization API : `NSSLOW_Init` causes the execution of the module's power-up self-tests. Upon successfully completing the power-up self-tests, `NSSLOW_Init` returns a non-NULL init context required by `NSSLOW_Shutdown` and `NSSLOWHASH_NewContext`.

Upon initialization of the cryptographic module library for the FIPS Approved mode of operation, the following power-up self-tests are performed by the module:

a) SHA-1 hash KAT,
b) SHA-256 hash KAT,

c) SHA-384 hash KAT,
d) SHA-512 hash KAT,
e) DSA signature verification integrity check

Restarting the NSS Freebl cryptographic module with the `NSSLOW_Shutdown` and `NSSLOW_Init` functions will cause the module to execute its power-up self-tests. This allows a user to execute these power-up self-tests on demand as defined in Section 4.9.1 of FIPS 140-2.

On success `NSSLOW_Init` will return an init context required by `NSSLOW_Shutdown` and `NSSLOWHASH_NewContext`. On failure `NSSLOW_Init` will return `NULL`.

In the FIPS Approved mode of operation the cryptographic, the module does not allow critical errors to compromise security. Whenever a critical error (e.g., a self-test failure) is encountered, the cryptographic module returns a `NULL` pointer in response to the intialization API (`NSSLOW_Init`) and prohibits all cryptographic services. Any subsequent attempt to create a new SHA context using `NSSLOWHASH_NewContext` while in the error state will fail and the module will return `NULL`.

In order to recover from an error state caused by a self-test failure condition, the module needs to be unloaded from the memory by terminating the application and then reloaded into memory by restarting the application, which should again initialize the module using `NSSLOW_Init`. This procedure will cause the module to be reinitialized in the FIPS-Approved mode and repeat all its power-up self-tests. To ensure that the module is operating in the FIPS-Approved mode, it should be verified that /proc/sys/crypto/fips_enabled contains the value “1”. If not, please follow the instructions in the Security Rules to appropriately initialize the module in the FIPS-Approved mode.

Specification of Services

Cryptographic module consists solely of public services which require no user authentication.

Service Category	Role	Function Name	Description	Cryptographic Keys and CSPs Accessed	Access type, RWZ
General purpose	User	NSSLOW_Init	Initializes the module library, checks for the FIPS Approved mode of operation. This function provides the power-up self-test service and the Show Status service.	Integrity Verification DSA public key	R
	User	NSSLOW_Shutdown	finalizes (shuts down) the module library	none	-
Message digesting	User	NSSLOWHASH_NewContext	Create a hashing context to be used by NSSLOWHASH_Begin, NSSLOWHASH_Update, NSSLOWHASH_End, NSSLOWHASH_Destroy, and NSSLOWHASH_Length	none	
	User	NSSLOWHASH_Begin	initializes a message-digesting operation	none	-
	User	NSSLOWHASH_Update	continues a multiple-part digesting operation	none	-
	User	NSSLOWHASH_End	finishes a multiple-part digesting operation and return result	none	-
	User	NSSLOWHASH_Destroy	Frees a hashing context created byt NSSLOWHASH_New Context	none	-
	User	NSSLOWHASH_Length	Returns the length of the selected hash	none	-

Sample Cryptographic Module Initialization Code

```
#include <stdio.h>
#include <nspr4/prtypes.h>
#include <nss3/hasht.h>
#include "nsslowhash.h"

  /* SHA-256 Known Digest Message (256-bits). */
    static const unsigned char sha256_known_digest[] = {
        0x38,0xa9,0xc1,0xf0,0x35,0xf6,0x5d,0x61,
        0x11,0xd4,0x0b,0xdc,0xce,0x35,0x14,0x8d,
        0xf2,0xdd,0xaf,0xaf,0xcf,0xb7,0x87,0xe9,
        0x96,0xa5,0xd2,0x83,0x62,0x46,0x56,0x79};

/* Known Hash Message (512-bits).  Used for all hashes (incl. SHA-N [N>1]). */
static const unsigned char known_hash_message[] = {
  "The test message for the MD2, MD5, and SHA-1 hashing algorithms." };



main(int argc, char **argv)
{
    NSSLOWInitContext *initCtx;
    NSSLOWHASHContext *ctx;
    unsigned char results_buf[64];
    int len;

   initCtx = NSSLOW_Init();
   if (initCtx == NULL) {
        printf("Couldn't init hash\n");
        return 1;
   }
   ctx = NSSLOWHASH_NewContext(initCtx, HASH_AlgSHA256);
   if (ctx == NULL) {
        printf("Couldn't get hash context\n");
        return 1;
   }
   NSSLOWHASH_Begin(ctx);
   NSSLOWHASH_Update(ctx, known_hash_message, 64);
   NSSLOWHASH_End(ctx, results_buf, &len, sizeof(results_buf));
   NSSLOWHASH_Destroy(ctx);
   NSSLOW_Shutdown(initCtx);

   if (len != sizeof(sha256_known_digest)) {
        printf("Hash result lengths do not match (%d != %d)\n",
                        len, sizeof(sha256_known_digest));
        return 2;
   }
   if (memcmp(sha256_known_digest, results_buf, len) != 0) {
        printf("Hash result not match\n");
        return 3;
   }
   printf(" hash completed and OK \n" );


   return 0;
}
```

Acknowledgments

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