NSS Cryptographic Module Freebl

Version 3.12.4

FIPS 140-2 Non-Proprietary Security Policy

Level 1 Validation

Red Hat, Inc.

Document Version 1.4

July 31 2009
## Table of Contents

- Introduction ........................................................................................................................................... 3
- Platform List ........................................................................................................................................ 3
- Security Rules .................................................................................................................................... 4
- Authentication Policy ............................................................................................................................ 6
  - Specification of Roles .......................................................................................................................... 6
  - Multiple Concurrent Operators ........................................................................................................ 6
- Module Ports and Interfaces .................................................................................................................... 7
  - Physical Cryptographic Boundary ......................................................................................................... 7
  - Logical Cryptographic Boundary ........................................................................................................ 8
  - Logical Interfaces ............................................................................................................................... 9
- Access Control Policy ............................................................................................................................. 9
- Security-Relevant Information ................................................................................................................ 9
  - Self-Tests ........................................................................................................................................... 9
  - Specification of Services ...................................................................................................................... 10
- Sample Cryptographic Module Initialization Code ................................................................................. 12
- Acknowledgments ............................................................................................................................... 13
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 glibcrypt library.

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, 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 IBM System x3550 running Red Hat Enterprise Linux v5
  - 32-bit binary on HP ProLiant DL145 running Red Hat Enterprise Linux v5

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 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. 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 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. Reactivation is accomplished by callingNSSLOW_Shutdown followed by NSSLOW_Init.

6. Resetting and restarting the NSS Freebl cryptographic module with the NSSLOW_Reset and NSSLOW_Init 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.

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:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cert#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHS</td>
<td>1048</td>
<td>SHA-1 (BYTE-only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA-256 (BYTE-only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA-384 (BYTE-only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHA-512 (BYTE-only)</td>
</tr>
<tr>
<td>DSA</td>
<td>366</td>
<td>DSA 1024 used internally for integrity checking</td>
</tr>
</tbody>
</table>

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

- 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 libraries/DLLs and the associated .chk files:
   - Red Hat Enterprise Linux 5 x86, and Red Hat Enterprise Linux 5 x86_64
     - libfreebl3.so
     - libfreebl3.chk

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 a directory specified in the following environment variable:

- 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 libfreebl3.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 libfreebl3.chk
```
Step 4: The kernel fip_enable flag must be set to '1'. The NSS Freebl cryptographic module detects this by reading /proc/sys/crypto/fips_enabled.

(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 'Specification of Services' table on page 11.

- 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 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, as the module does not implement any 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

NSSLOW / NSSLOWHASH Interface

libfreebl3.so
libsfreebl3.chk

Cryptographic Boundary

OS
Logical Interfaces

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 does not access CPSs.

Security-Relevant Information

Self-Tests

In the FIPS Approved mode of operation the cryptographic module does not allow critical errors to compromise security. Whenever a critical error (e.g., a self-test failure) is encountered, the cryptographic module enters an error state and the library needs to be reinitialized to resume normal operation. Reinitialization is accomplished by calling NSSLOW_Shutdown followed by NSSLOW_Init.
Upon initialization of the cryptographic module library for the FIPS Approved mode of operation, the following power-up self-tests shall be performed:

a) SHA-1 hash,
b) SHA-256 hash,
c) SHA-384 hash,
d) SHA-512 hash,
e) DSA signature verification integrity check

Resetting and restarting the NSS cryptographic module with the NSSLOW_Reset and NSSLOW_Init 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.

On success NSSLOW_Init will return an init context required by NSSLOW_Reset, NSSLOW_Shutdown and NSSLOWHASH_CreateContext. On failure NSSLOW_Init will return NULL.

**Specification of Services**

Cryptographic module consists solely of public services which require no user authentication.
<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>General purpose</td>
<td>User</td>
<td>NSSLOW_Init</td>
<td>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.</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>User</td>
<td></td>
<td>NSSLOW_Shutdown</td>
<td>finalizes (shuts down) the module library</td>
<td>none</td>
<td>Z</td>
</tr>
<tr>
<td>User</td>
<td></td>
<td>NSSLOW_Reset</td>
<td>resets the power on self-test flag</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>Message digesting</td>
<td>User</td>
<td>NSSLOWHASH_NewContext</td>
<td>Create a hashing context to be used by NSSLOWHASH_Begin, NSSLOWHASH_Update, NSSLOWHASH_Destroy, and NSSLOWHASH_Length</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>User</td>
<td></td>
<td>NSSLOWHASH_Begin</td>
<td>initializes a message-digesting operation</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>User</td>
<td></td>
<td>NSSLOWHASH_Update</td>
<td>continues a multiple-part digesting operation</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>User</td>
<td></td>
<td>NSSLOWHASH_Destroy</td>
<td>finishes a multiple-part digesting operation</td>
<td>none</td>
<td>-</td>
</tr>
<tr>
<td>User</td>
<td></td>
<td>NSSLOWHASH_Length</td>
<td>Returns the length of the selected hash</td>
<td>none</td>
<td>-</td>
</tr>
</tbody>
</table>
Sample Cryptographic Module Initialization Code

```c
#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.