Hewlett Packard Enterprise

FIPS 140-2 Non-Proprietary Security Policy

Hewlett Packard Enterprise NSS Crypto Module

Software Version 4.0

Document Version 1.4

April 4, 2018

Prepared For:

Hewlett Packard Enterprise

Hewlett Packard Enterprise
3000 Hanover St
Palo Alto, CA 94304
hpe.com

Prepared By:

SafeLogic Inc.
530 Lytton Ave, Suite 200
Palo Alto, CA 94301
www.safelogic.com

Abstract

This document provides a non-proprietary FIPS 140-2 Security Policy for Hewlett Packard Enterprise NSS Crypto Module.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 About FIPS 140

Federal Information Processing Standards Publication 140-2 — Security Requirements for Cryptographic Modules specifies requirements for cryptographic modules to be deployed in a Sensitive but Unclassified environment. The National Institute of Standards and Technology (NIST) and Communications Security Establishment Canada (CSE) Cryptographic Module Validation Program (CMVP) run the FIPS 140 program. The NVLAP accredits independent testing labs to perform FIPS 140 testing; the CMVP validates modules meeting FIPS 140 validation. *Validated* is the term given to a module that is documented and tested against the FIPS 140 criteria.

More information is available on the CMVP website at http://csrc.nist.gov/groups/STM/cmvp/index.html.

1.2 About this Document

This non-proprietary Cryptographic Module Security Policy for Hewlett Packard Enterprise NSS Crypto Module provides an overview of the product and a high-level description of how it meets the overall Level 1 security requirements of FIPS 140-2.

Hewlett Packard Enterprise NSS Crypto Module may also be referred to as the “module” in this document.

1.3 External Resources

The HPE website (https://www.hpe.com) contains information on HPE services and products. The Cryptographic Module Validation Program website contains links to the FIPS 140-2 certificate and HPE contact information.

1.4 Notices

This document may be freely reproduced and distributed in its entirety without modification.

2 Hewlett Packard Enterprise NSS Crypto Module

2.1 Cryptographic Module Specification

Hewlett Packard Enterprise NSS Crypto Module (hereafter referred to as the "Module") is a software library supporting FIPS 140-2 approved cryptographic algorithms. The software version is 4.0. For the purposes of the FIPS 140-2 validation, its embodiment type is defined as multi-chip standalone. The Module is an open-source, general-purpose cryptographic library, with an API based on the industry standard PKCS #11 version 2.20. It combines a vertical stack of Linux components intended to limit the external interface each separate component may provide.

The module relies on the physical characteristics of the host platform. The module's physical cryptographic boundary is defined by the enclosure of the host platform.

All operations of the module occur via calls from host applications and their respective internal daemons/processes. As such there are no untrusted services calling the services of the module.

The module's logical cryptographic boundary is the shared library of files and their integrity check signature files which are delivered through Red Hat Package Manager (RPM) as listed below:

- Nss-softokn RPM file with version 3.16.2.3-13, which contains the following files:
  - `/usr/lib64/libnssdbm3.chk (64 bits)`
  - `/usr/lib64/libnssdbm3.so (64 bits)`
  - `/usr/lib64/libsoftokn3.chk (32 bits)`
  - `/usr/lib64/libsoftokn3.so (32 bits)`
  - `/usr/lib.libnssdbm3. chk (32bits)`
  - `/usr/lib.libnssdbm3. so (32bits)`
  - `/usr/lib/libsoftokn3.chk (32 bits)`
  - `/usr/lib/libsoftokn3.so (32 bits)`
- Nss-softokn-freebl RPM file with version 3.16.2.23-13, which contains the following files:
  - `/lib64/libfreeblpriv3.chk (64 bits)`
  - `/lib64/libfreeblpriv3.so (64 bits)`
  - `/lib64/libfreeblpriv3.chk (32 bits)`
  - `/lib64/libfreeblpriv3.so (32 bits)`

2.1.1 Validation Level Detail

The following table lists the level of validation for each area in FIPS 140-2:

FIPS 140-2 Section Title	Validation Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	2
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
Electromagnetic Interference / Electromagnetic Compatibility	1
Self-Tests	1
Design Assurance	2
Mitigation of Other Attacks	1

Table 1 – Validation Level by FIPS 140-2 Section

2.1.2 Modes of Operation

The module supports two modes of operation: Approved and Non-approved. The module will be in FIPS-approved mode when all power up self-tests have completed successfully and only Approved algorithms are invoked. See Table 2 below for a list of the supported Approved algorithms and Table 3 for allowed algorithms. The non-Approved mode is entered when a non-Approved algorithm is invoked. See Table 4 for a list of non-Approved algorithms.

2.1.3 Approved Cryptographic Algorithms

The module's cryptographic algorithm implementations have received the following certificate numbers from the Cryptographic Algorithm Validation Program.

CAVP Cert.	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
3605, 3606, 3607, 3609	**AES**	FIPS 197 SP 800-38A	ECB, CBC and CTR	128, 192, 256	Encryption, Decryption
3605, 3606, 3607, 3609	**GCM/GMAC**	SP 800-38D	AES	128, 192, 256	Decryption, Authentication
936, 937	**DRBG**	SP 800-90A	Hash DRBG	256	Random Bit Generation The module generates keys whose strengths are modified by available entropy.
1002, 1003	**DSA**[1]	FIPS 186-4	PQG Verification, Key Pair Generation, Signature Generation, Signature Verification	1024, 2048, 3072 bits (1024 only for SigVer)	Digital Signature Services

[1] DSA signature generation with SHA-1 is only for use with protocols.

CAVP Cert.	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
739, 740	**ECDSA**	FIPS 186-4	Key Pair Generation, Signature Generation, Signature Verification, Public Key Validation	P-256, P-384, P- 521,	Digital Signature Services
2300, 2301	**HMAC**	FIPS 198-1	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512		Generation, Authentication
626, 627	**KDF, Existing Application-Specific**[2]	SP 800-135	TLS v1.0/1.1 KDF, TLS 1.2 KDF		TLS pre-master secret and master secret
1854, 1855, 2034, 2035	**RSA**	FIPS 186-4 PKCS #1 v1.5		2048, 3072 bits (1024 only for SigVer)	Signature Generation, Signature Verification
2966, 2967	**SHA**	FIPS 180-4	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512		Digital Signature Generation, Digital Signature Verification, non-Digital Signature Applications

[2] These protocols have not been reviewed or tested by the CAVP and CMVP.

CAVP Cert.	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
2007, 2008	**Triple-DES**	SP 800-67	TECB, TCBC and CTR	2-key, 3-key	Decryption

Table 2 – FIPS-Approved Algorithm Certificates

Note: The TLS protocol has not been reviewed or tested by the CAVP and CMVP.

2.1.4 Non-Approved but Allowed Cryptographic Algorithms

The module supports the following non-FIPS 140-2 approved but allowed algorithms that may be used in the Approved mode of operation.

Algorithm	Use
Diffie-Hellman Key Agreement	[IG D.8] Diffie-Hellman 2048-bit and 15360 bit key agreement primitive for use with system-level key establishment; not used by the module to establish keys within the module (key agreement; key establishment methodology provides between 112 and 256 bits of encryption strength)
Elliptic Curve Diffie-Hellman	Elliptic Curve Diffie-Hellman using P-256, P-384 and P-521 curves is used for system-level key establishment; not used by the module to establish keys within the module (key agreement; key establishment methodology provides between 128 and 256 bits of encryption strength)
MD5 within TLS	[IG D.2]
Non-SP 800-56B compliant RSA Key Transport	[IG D.9] RSA may be used by a calling application as part of a key encapsulation scheme. Allowed through December 31st, 2017 per IG D.9. Key sizes: 2048 bits or larger Key wrapping; key establishment methodology provides between 112 and 256 bits of encryption strength.

Table 3 – Non-Approved but Allowed Cryptographic Algorithms

2.1.5 Non-Approved Mode of Operation

The module supports a non-approved mode of operation. The algorithms listed in this section are not to be used by the operator in the FIPS Approved mode of operation.

Algorithm	Use
AES- GCM (non-compliant)	Encryption
AES CTS	Encryption, Decryption
AES Key Wrapping	SP 800-38F Key Wrapping
Camellia	Encryption, Decryption
DES	Encryption, Decryption

Algorithm	Use
Diffie-Hellman	Key Agreement
DSA (non-compliant[3])	Public Key Cryptography
ECDSA (non-compliant[4])	Public Key Cryptography
J-PAKE	Key Agreement
MD2	Hashing
MD5	Hashing
RC2	Encryption, Decryption
RC4	Encryption, Decryption
RC5	Encryption, Decryption
RSA (non-compliant[5])	Public Key Cryptography
SEED	Encryption, Decryption
Triple-DES with 2-key	Encryption
Tripe-DES Key Wrapping	SP 800-38F Key Wrapping

Table 4 – Non-Approved Cryptographic Functions for use in non-FIPS mode only

[3] Deterministic signature calculation, support for additional digests, and key sizes.
[4] Deterministic signature calculation, support for additional digests, and key sizes.
[5] Support for RSA key pair generation, support for additional key sizes.

2.2 Critical Security Parameters and Public Keys

2.2.1 Critical Security Parameters

The table below provides a complete list of Critical Security Parameters used within the module:

CSP	Generation	Storage	Entry/Output	Zeroization
AES 123, 192, and 256 bit keys	Use of NIST SP800-90A DRBG	Application memory or key database	Encrypted through key wrapping using FC_WrapKey (in non-Approved mode)	Automatically zeroized when freeing the cipher handle
Triple-DES 168 bits	Use of NIST SP800-90A DRBG	Application memory or key database	Encrypted through key wrapping using FC_WrapKey (in non-Approved mode)	Automatically zeroized when freeing the cipher handle
DSA 2048 and 3072 bit private keys	Use of NIST SP800-90A DRBG in DSA key generation mechanism	Application memory or key database	Encrypted through key wrapping using FC_WrapKey (in non-Approved mode)	Automatically zeroized when freeing the cipher handle
ECDSA private keys based on P-256, P-384 and P-521	N/A (supplied by the calling application)	Application memory or key database	Encrypted through key wrapping using FC_WrapKey (in non-Approved mode)	Automatically zeroized when freeing the cipher handle
RSA 2048 and 3072 bit private keys	N/A (supplied by the calling application)	Application memory or key database	Encrypted through key wrapping using FC_WrapKey (in non-Approved mode)	Automatically zeroized when freeing the cipher handle
HMAC keys with at least 112 bits	Use of NIST SP800-90A DRBG	Application memory or key database	Encrypted through key wrapping using FC_WrapKey (in non-Approved mode)	Automatically zeroized when freeing the cipher handle
DRBG entropy input string and seed	Obtained from /dev/urandom	Application memory	N/A	Automatically zeroized when freeing the DRBG handle
DRBG V and C values	Derived from the entropy input string as defined in NIST SP800-90A	Application memory	N/A	Automatically zeroized when freeing the DRBG handle

CSP	Generation	Storage	Entry/Output	Zeroization
TLS pre-master secret	Use of NIST SP800-90A DRBG in Diffie-Hellman or EC Diffie-Hellman key agreement scheme	Application memory	N/A	Automatically zeroized when freeing the cipher handle
TLS master secret	Derived from TLS pre-master secret by using key derivation	Application memory	N/A	Automatically zeroized when freeing the cipher handle
Diffie-Hellman private components with size between 2048 bits and 15360 bits	Use of NIST SP800-90ADRBG in Diffie-Hellman key agreement scheme	Application memory	N/A	Automatically zeroized when freeing the cipher handle
EC Diffie-Hellman private components based on P-256, P-384 and P-521 curves	Use of NIST SP800-90A DRBG in EC Diffie-Hellman key agreement scheme	Application memory	N/A	Automatically zeroized when freeing the cipher handle
User Passwords	N/A (supplied by the calling application)	Application memory or key database in salted form	N/A (input through API parameter)	Automatically zeroized when the module is re-initialized or overwritten when the user changes its password

Table 5 – Critical Security Parameters

Note: The /dev/urandom is an NDRNG located within the module's physical boundary but outside the logical boundary.

2.2.2 Random Number Generation

The Module employs a NIST SP800-90A Hash_DRBG with SHA-256 as a random number generator. The random number generator is seeded by obtaining random data from the operating system via /dev/urandom. The entropy source /dev/urandom provides at least 880 bits of random data available to the Module to obtain.

Reseeding is performed by pulling more data from /dev/urandom. A product using the Module should periodically reseed the module's random number generator with unpredictable noise by calling FC_SeedRandom. After 2^{48} calls to the random number generator the Module reseeds automatically.

The Module performs DRBG health testing as specified in section 11.3 of NIST SP800-90A. The Module provides at least 112 bits of entropy.

2.2.3 Key/CSP Storage

The Module employs the cryptographic keys and CSPs in the FIPS Approved mode operation as listed in Table 5 – Critical Security Parameters. The module does not perform persistent storage for any keys or CSPs. Note that the private key database (provided with the files key3.db/key4.db) mentioned in Table 5 is within the Module's physical boundary but outside its logical boundary.

2.2.4 Key/CSP Zeroization

The application that used the Module is responsible for appropriate zeroization of the key material. The Module provides zeroization methods to clear the memory region previously occupied by a plaintext secret key, private key or password. A plaintext secret and private keys must be zeroized when it is passed to a FC_DestroyObject call. All plaintext secret and private keys must be zeroized when the Module is shut down (with a FC_Finalize call), reinitialized (with a FC_InitToken call), or when the session is closed (with a FC_Close Session or FC_CloseAllSessions call). All zeroization is to be performed by storing the value 0 into every byte of memory region that is previously occupied by a plaintext secret key, private key or password.

Zeroization is performed in a time that is not sufficient to compromise plaintext secret or private keys and password.

2.3 Module Interfaces

The figure below shows the module's physical and logical block diagram:

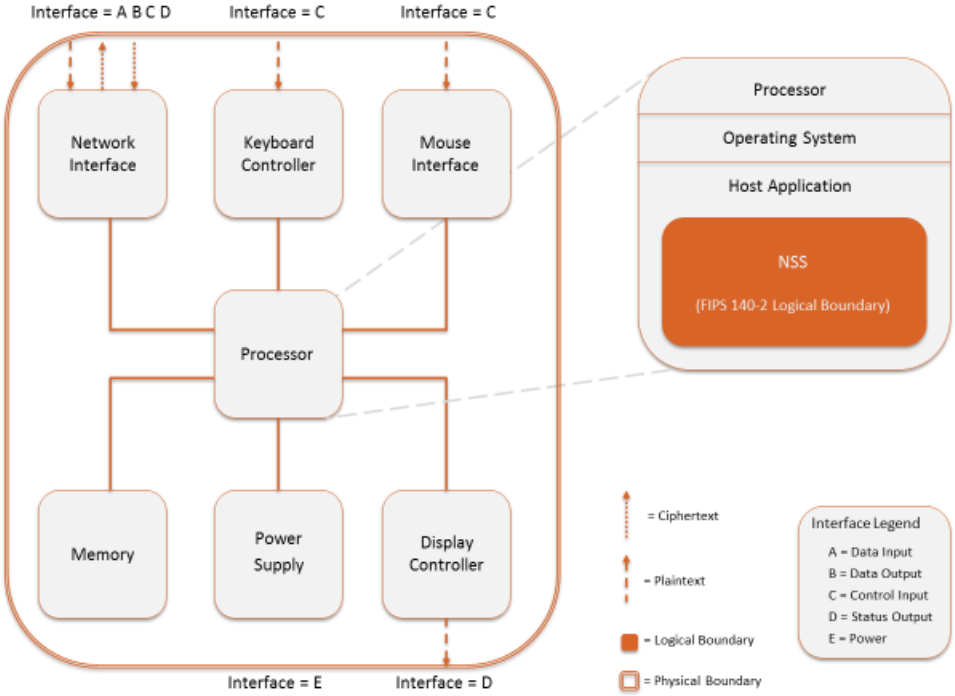

Figure 1 – Module Boundary and Interfaces Diagram

The interfaces (ports) for the physical boundary include the computer keyboard port, mouse port, network port, USB ports, display and power plug. When operational, the module does not transmit any information across these physical ports because it is a software cryptographic module. Therefore, the module's interfaces are purely logical and are provided through the Application Programming Interface (API) following the PKCS #11 specification, the database files in a kernel file system, the environment variables and configuration files. The logical interfaces expose services that applications directly call, and the API provides functions that may be called by a referencing application (see Section 2.4 – Roles, Services, and Authentication for the list of available functions). The module distinguishes between logical interfaces by logically separating the information according to the defined API.

The API provided by the module is mapped onto the FIPS 140- 2 logical interfaces: data input, data output, control input, and status output. Each of the FIPS 140- 2 logical interfaces relates to the module's callable interface, as follows:

FIPS 140-2 Interface	Logical Interface	Module Physical Interface
Data Input	API input parameters – plaintext and/or ciphertext data	Network Interface
Data Output	API output parameters and return values – plaintext and/or ciphertext data	Network Interface
Control Input	API method calls- method calls, or input parameters, that specify commands and/or control data used to control the operation of the module (/proc/sys/crypto/fips_enabled)	Keyboard Interface, Mouse Interface
Status Output	API output parameters and return/error codes that provide status information used to indicate the state of the module	Display Controller, Network Interface
Power	None	Power Supply

Table 6 – Logical Interface / Physical Interface Mapping

As shown in Figure 1 – Module Boundary and Interfaces Diagram and Table 6 – Logical Interface / Physical Interface Mapping, the output data path is provided by the data interfaces and is logically disconnected from processes performing key generation or zeroization. No key information will be output through the data output interface when the module zeroizes keys.

The Module does not use the same buffer for input and output. After the 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 can be inadvertently leaked.

The logical interfaces of the Module consist of the PKCS #11 (Cryptoki) API. The API itself defines the Module's logical boundary, i.e, all access to the Module is through this API. The functions in the PKCS #11 API are listed in .

2.3.1 Inhibition of Data Output

All data output via the data output interface is inhibited when the NSS cryptographic module is performing self-tests or in the Error state.

During self-tests: All data output via the data output interface is inhibited while self-tests are executed.

In Error state: The Boolean state variable sftk fatalError tracks whether the NSS cryptographic module is in the Error state. Most PKCS #11 functions, including all the functions that output data via the data output interface, check the sftk_fatalError state variable and, if it is true, return the CKR_DEVICE_ERROR error code immediately. Only the functions that shut down and

restart the module, reinitialize the module, or output status information can be invoked in the Error state.

These functions are FC_GetFunctionList, FC_Initialize, FC_Finalize, FC_GetInfo, FC_GetSlotList, FC_GetSlotinfo, FC_GetTokeninfo, FC_InitToken, FC_CloseSession, FC_CloseAllSessions, and FC WaitForSlotEvent.

2.3.2 Disconnecting the output Data Path from the Key Process

During key generation and key zeroization, the Module may perform audit logging, but the audit records do not contain sensitive information. The Module does not return the function output arguments until the key generation or key zeroization is finished. Therefore, the logical paths used by output data exiting the module are logically disconnected from the processes/threads performing key generation and key zeroization.

2.4 Roles, Services, and Authentication

2.4.1 Roles

The module supports two distinct operator roles, User and Crypto Officer (CO). The cryptographic module implicitly maps the two roles to the services. A user is considered the owner of the thread that instantiates the module and, therefore, only one concurrent user is allowed.

The module does not support a Maintenance role or bypass capability. The module does not support authentication.

Role	Role Description	Authentication Type
CO	Crypto Officer – Installs and initializes the module. The CO can access other general-purpose services (such as message digest and random number generation services) and status services of the Module. The CO does have access to any service that utilizes the secret and private keys of the Module. The CO must control the access to the Module both before and after installation, including the management of physical access to the computer, executing the Module code as well as management of the security facilities provided by the operating system.	Role Based Authentication

Role	Role Description	Authentication Type
User	User – The User role had access to all cryptography secure services which use the secret or private keys of the Module. It is also responsible for the retrieval, updating and deletion of keys from the private key database.	Role Based Authentication

Table 7 – Description of Roles

2.4.2 Assumption of Roles

The CO role is implicitly assumed by an operator while installing the Module by following the instructions in Section 3.1 and while performing other CO services on the Module.
The Module implements a password-based authentication for the User role. To perform any security services under the User role, an operator must log into the Module and complete an authentication procedure using the password information unique to the User role operator.

The password is passed to the Module via the API function as an input argument and will not be displayed. The return value of the function is the only feedback mechanism, which does not provide any information that could be used to guess or determine the User's password. The password is initialized by the CO role as part of the module initialization and can be changed by the User role operator.

If a User role service is called before the operator is authenticated, it returns the CKR_USER_NOT_LOGGED_IN error code. The operator must call the FC_Login function to provide the required authentication.

Once a password has been established for the Module, the user is allowed to use the security services if and only if the user is successfully authenticated to the Module. Password establishment and authentication are required for the operation of the Module. When Module is powered off, the result of previous authentication will be cleared and the user needs to be re-authenticated.

2.4.3 Strength of Authentication Mechanism

The Module imposes he 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 (a-z), ASCII uppercase (A-Z), ASCII non-alphanumeric characters (space and other ASCII special characters such as '$' and '!'), and non ASCII characters (Latin characters such as 'é', 'ß'; Greek characters such as 'Ω', 'θ', other non-ASCII special characters such as '¿'). 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 maximum probability that a random guess of the password will succeed, we assume that:

- The characters of the password are independent with each other.
- The password contains the smallest combination of the characters classes, which is give digits, one ASCII lowercase letter and one ASCII uppercase letter. The probability to guess every character successfully is $(1/10)^5*(1/26)*(1/26) = 1/67,600,000$.

Since the password can contain seven characters from any three or more of the aforementioned give character classes, the probability that a random guess of the password will succeed is less than or equals to 1/67,600,000, which is smaller than the required threshold 1/1,000,000.

After each failed authentication attempt in the FIPS Approved mode, the Hewlett Packard Enterprise NSS Crypto 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 or equal to 60*1/67,600,000 = 0.089* (1/100,000), which is smaller than the required threshold 1/100,000.

2.4.4 Services

The Module has a set of API functions denoted by FC_xxx. All the API functions are listed in Table 8 – Module Services.

Among the module's API functions, only FC_GetFunctionlist is exported and therefore callable by its name. All the other API functions must be called via the function pointers returned by FC_GetFunctionlist. It returns a CK_FUNCTION_LIST structure containing function pointers named C_xxx such as (_Initialize and (_Finalize. The C_xxx function pointers in the CK_FUNCTION_LIST structure returned by FC_GetFunctionlist point to the FC_xxx funct ions .

The following convention is used to describe API function calls. Here FC_Initialize is used as examples:

- When "call FC_Initialize" is mentioned, the technical equivalent of "call the FC_Initialize function via the (_Initialize function pointer in the CK_FUNCTION_LIST structure returned by
  FC_GetFunctionlist" is implied.

The Module supports Crypto-Officer services which require no operator authentication, and User services which require operator authentication. Crypto-Officer services do not require access to the secret and private keys and other CSPs associated with the user. The message digesting services are available to Crypto-Officer only when CSPs are not accessed. User services which access CSPs (e.g., FC_GenerateKey, FC_GenerateKeyPair) require operator authentication.

All services implemented by the module are listed in Table 8 – Module Services. The third column provides a description of each service and availability to the Crypto Officer and User, in columns 4 and 5, respectively.

Note: The message digesting functions (except FC_DigestKey) that do not use any keys of the Module can be accessed by the Crypto-Officer role and do not require authentication to the Module. The FC_DigestKey API function computes the message digest (hash) of the value of a secret key, so it is available only to the User role.

Service	Function	Description	CO	User
Get the Function List	FC_GetFunctionList	Return a pointer to the list of function pointers for the operational mode	X	
Module Initialization	FC_InitToken	Initialize or re-initialize a token	X	
	FC_InitPIN	Initialize the user's password, i.e., set the user's initial password		
General Purpose	FC_Initialize	Initialize the module library	X	
	FC_Finalize	Finalize (shut down) the module library		
	FC_GetInfo	Obtain general information about the module library		
Slot and Token Management	FC_GetSLotList	Obtain a list of slots in the system	X	
	FC_GetSlotInfo	Obtain information about a particular slot		
	FC_GetTokenInfo	Obtain information about the token (This function provides the Show Status service)		
	FC_GetMechanismList	Obtain a list of mechanisms (cryptographic algorithms) supported by a token		
	FC_GetMechanismInfo	Obtain information about a particular mechanism		
	FC_SetPIN	Change the user's password		X

Service	Function	Description	CO	User
Session Management	FC_OpenSession	Open a connection (session) between an application and a particular token	X	
	FC_CloseSession	Close a session		
	FC_CloseAllSessions	Close all sessions with a token		
	FC_GetSessionInfo	Obtain information the session (This function provides the Show Status service)		
	FC_GetOperationState	Save the state of the cryptographic operations in a session (This function is only implemented for message digest operations)		
	FC_SetOperationState	Restore the state of the cryptographic operations in a session (This function is only implemented for message digest operations		
	FC_Login	Log into a token		X
	FC_Logout	Log out from a token		
Object Management	FC_CreateObject	Creat a new object	X	
	FC_CopyObject	Create a copy of an object		
	FC_DestroyObject	Destroy an object		
	FC_GetObjectSize	Obtain the size of an object in bytes		
	FC_GetAttributeValue	Obtain an attribute value of an object		
	FC_SetAttributeValue	Modify an attribute value of an object		
	FC_FindObjectsInit	Initialize an object search operation		
	FC_FindObjects	Continue an object search operation		
	FC_FindObjectsFinal	Finish an object search operation		

Service	Function	Description	CO	User
Encryption and Decryption	FC_EncryptInit	Initialize an encryption operation		X
	FC_Encrypt	Encrypt single-part data		
	FC_EncryptUpdate	Continue a multiple-part encryption operation		
	FC_EncryptFinal	Finish a multiple-part encryption operation		
	FC_DecryptInit	Initialize a decryption operation		
	FC_Decrypt	Decrypt single-part encrypted data		
	FC_DecryptUpdate	Continue a multiple-part decryption operation		
	FC_DecryptFinal	Finish a multiple-part decryption operation		
Message Digest	FC_DigestInit	Initialize a message-digesting operation	X	
	FC_Digest	Digest single-part data		
	FC_DigestUpdate	Continue a multiple-part digesting operation		
	FC_DigestFinal	Finish a multiple-part digesting operation		
	FC_DigestKey	Continue a multiple-part message-digestion operation by digesting the value of a secret key as part of the data already digested		X

Service	Function	Description	CO	User
Signature Generation and Verification	FC_SignInit	Initialize a signature operation		X
	FC_Sign	Sign single part data		
	FC_SignUpdate	Continue a multiple-part signature operation		
	FC_SignFinal	Finish a multiple-part signature operation		
	FC_SignRecoverInit	Initialize a signature operation, where the data can be recovered from the signature		
	FC_SignRecover	Sign single-part data, where the data can be recovered from the signature		
	FC_VerifyInit	Initialize a verification operation		
	FC_Verify	Verify a signature on single-part data		
	FC_VerifyUpdate	Continue a multiple-part verification operation		
	FC_VerifyFinal	Finish a multiple-part verification operation		
	FC_VerifyRecoverInit	Initialize a verification operation, where the data is recovered from the signature		
	FC_VerifyRecover	Verify a signature on single-part data, where the data is recovered from the signature		

Service	Function	Description	CO	User
Dual-function Cryptographic Operations	FC_DigestEncryptUPdate	Continue a multiple-part digesting and encryption operation		X
	FC_DecryptDigestUpdate	Continue a multiple-part decryption and digesting operation		
	FC_SignEncryptUpdate	Continue a multiple-part signing and encryption operation		
	FC_DecryptVerifyUpdate	Continue a multiple-part decryption and verify operation		
Key Management	FC_GenerateKey	Generate a secret key (Also used by TLS to generate a pre-master secret)		X
	FC_GenerateKeyPair	Generate a public/private key pair (This function performs the pair-wise consistency tests)		
	FC_DeriveKey	Derive a key from TLS master secret which is derived from TLS pre-master secret		
Random Number Generation	FC_SeedRandom	Mix in additional seed material to the random number generator	X	
	FC_GenerateRandom	Generate random data (This function performs the continuous random generator test)		
Parallel Function Management	FC_GetFunctionStatus	A legacy function, which simply returns the value 0x00000051 (function not parallel)	X	
	FC_CalcelFunction	A legacy function, which simply returns the value 0x00000051 (function not parallel)		
Self Tests	N/A	The self tests are performed automatically when loading the module	X	

Service	Function	Description	CO	User
Zeroization	FC_InitToken FC_Finalize FC_CloseSession FC_CloseAllSessions	All CSPs are automatically zeroized when freeing the cipher handle	X	
	FC_DestroyObjects			X

Table 8 – Module Services

Table 9 lists all the services available in non-Approved mode with API function and the non-Approved algorithm that the function may invoke. Please note that the functions are the same as the ones listed in Table 8, but the underneath non-Approved algorithms are invoked. Please also refer to Table 4 for the non-Approved algorithms. If any service invokes the non-Approved algorithms, then the module will enter non-Approved mode implicitly.

Service	Function	Non-Approved Algorithm Invoked
Encryption and Decryption	FC_EncryptInit FC_Encrypt FC_EncryptUpdate FC_EncryptFinal	AES GCM mode, AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
	FC_DecryptInit FC_Decrypt FC_DecryptUpdate FC_DecryptFinal	AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED
Message Digest	FC_DigestInit FC_DigestUpdate FC_DigestKey FC_DigestFinal	MD2, MD5

Service	Function	Non-Approved Algorithm Invoked
Signature Generation and Verification	FC_SignInit FC_Sign FC_SignUpdate FC_SignFinal FC_SignRecoverInit FC_SignRecover	DSA signature generation with non-compliant key size, RSA signature generation with non-compliant key size
	FC_VerifyInit FC_Verify FC_VerificyUpdate FC_VerifyFinal FC_VerifyRecoverInit FC_VerifyRecover	DSA signature verification with non-compliant key size, RSA signature verification with non-compliant key size
Dual-function Cryptographic Operations	FC_DigestEncryptUpdate	MD2, MD5, AES GCM mode, AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
	FC_DecryptDigestUpdate	AES STS mode, Camellia, DES RC2, RC4, RC5, SEED, MD2, MD5
	FC_SignEncryptUpdate	DSA signature generation with non-compliant key size, RSA signature generation with non-compliant key size, AES GCM mode, AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, Two-key Triple-DES
	FC_DecryptVerifyUpdate	AES CTS mode, Camellia, DES, RC2, RC4, RC5, SEED, DSA signature verification with non-compliant key size, RSA signature verification with non-compliant key size
Key Management	FC_GenerationKeyPair	DSA domain parameter generation, DSA domain parameter verification with non-compliant key size, DSA key pair generation with non-compliant key size, RSA key pair generation

Service	Function	Non-Approved Algorithm Invoked
	FC_KeyWrapKey	AES key wrapping (encrypt) based on NIST SP800-38F, Triple-DES key wrapping (encrypt) using Two-key Triple-DES, RSA key wrapping (encrypt) with non-compliant key size
	FC_UnwrapKey	AES key wrapping (encrypt) based on NIST SP800-38F, Triple-DES key wrapping (decrypt) using Two-key Triple-DES, RSA key wrapping (decrypt) with non-compliant key size
	FC_DeriveKey	Diffie-Hellman key agreement with non-compliant key size, J-PAKE key agreement

Table 9 – Services in Non-Approved Mode

Table 10 – CSP Access Rights within Services defines the relationship between access to CSPs and the different module services. The modes of access shown in the table are defined as:

G = Generate: The module generates the CSP.

R = Read: The module reads the CSP. The read access is typically performed before the module uses the CSP.

E = Execute: The module executes using the CSP.

W = Write: The module writes the CSP. The write access is typically performed after a CSP is imported into the module, when the module generates a CSP, or when the module overwrites an existing CSP.

Z = Zeroize: The module zeroizes the CSP.

Service	CSPs												
	AES keys	Triple-DES bit keys	DSA private keys	ECDSA private keys	RSA Private Keys	HMAC Keys	DRBG entropy input string and seed	DRBG V and C values	TLS pre-master secret	TLS Master Secret	Diffie-Hellman private components	EC Diffie-Hellman private components	User Passwords
Get the Function List	-	-	-	-	-	-	-	-	-	-	-	-	-
Module Initialization	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	ZW
General Purpose	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z

Service	CSPs												
	AES keys	Triple-DES bit keys	DSA private keys	ECDSA private keys	RSA Private Keys	HMAC Keys	DRBG entropy input string and seed	DRBG V and C values	TLS pre-master secret	TLS Master Secret	Diffie-Hellman private components	EC Diffie-Hellman private components	User Passwords
Slot and Token Management	-	-	-	-	-	-	-	-	-	-	-	-	RW
Session Management	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	ZR
Object Management	RWZ	RWZ	RWZ	RWZ	RWZ	RWZ	RWZ	RWZ	RWZ	RWZ	RWZ	RWZ	RWZ
Encryption and Decryption	R	R	-	-	-	-	-	-	-	-	-	-	-
Message Digest	-	-	-	-	-	R	-	-	-	-	-	-	-
Signature Generation and Verification	-	-	R	R	R	R	-	-	-	-	-	-	-
Dual-function Cryptographic Operations	R	R	R	R	R	R	-	-	-	-	-	-	-
Key Management	W	W	-	-	-	W	-	-	RW	-	W	W	-
Random Number Generation	-	-	-	-	-	-	RW	RW	-	-	-	-	-

Service	CSPs												
	AES keys	Triple-DES bit keys	DSA private keys	ECDSA private keys	RSA Private Keys	HMAC Keys	DRBG entropy input string and seed	DRBG V and C values	TLS pre-master secret	TLS Master Secret	Diffie-Hellman private components	EC Diffie-Hellman private components	User Passwords
Parallel Function Management	-	-	-	-	-	-	-	-	-	-	-	-	-
Self Tests	-	-	R	-	-	-	-	-	-	-	-	-	-
Zeroization	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z

Table 10 – CSP Access Rights within Services

2.5 Physical Security

The module is a software-only module and does not have physical security mechanisms.

2.6 Operational Environment

The module operates in a modifiable operational environment under the FIPS 140-2 definitions.

The module runs on a GPC running one of the operating systems specified in the approved operational environment list in this section. Each approved operating system manages processes and threads in a logically separated manner. The module's user is considered the owner of the calling application that instantiates the module.

In FIPS Approved mode, the ptrace system call, the debugger gdb, and the strace shall not be used. In addition, other tracing mechanisms offered by the Linux environment, such as ftrace or styemtap, shall not be used.

The module was tested on the following platforms:

Hardware	Processor	Operating System	w/AES-NI	Without AES-NI
HP Proliant DL380p Gen8	Intel® Xeon® E5- 2600 v2 product family	Red Hat Enterprise Linux 7.1	Yes	Yes

Table 11 – FIPS Tested Configurations

Compliance is maintained for other versions of the respective operating system family where the binary is unchanged. No claim can be made as to the correct operation of the module or the security strengths of the generated keys when ported to an operational environment which is not listed on the validation certificate.

The GPC(s) used during testing met Federal Communications Commission (FCC) FCC Electromagnetic Interference (EMI) and Electromagnetic Compatibility (EMC) requirements for business use as defined by 47 Code of Federal Regulations, Part15, Subpart B. FIPS 140-2 validation compliance is maintained when the module is operated on other versions of the GPOS running in single user mode, assuming that the requirements outlined in NIST IG G.5 are met.

2.7 Self-Tests

Each time the module is powered up, it tests that the cryptographic algorithms still operate correctly and that sensitive data have not been damaged. Power-up self–tests are available on demand by power cycling the module.

On power-up or reset, the module performs the self-tests that are described in Table 12 – Power-Up Self-Tests

All KATs must be completed successfully prior to any other use of cryptography by the module. If one of the KATs fails, the module enters the Self-Test Failure error state. The Module returns the error code CKR_DEVICE_ERROR to the calling application to indicate the Error state. The Module needs to be reinitialized in order to recover from Error state.

2.7.1 Power-Up Self-Tests

Test Target	Description
AES	KATs for ECB and CB modes: encryption and decryption are tested separately
Triple-DES	KATs for ECB and CBC modes: encryption and decryption are tested separately
DSA	KAT: signature generation and verification are tested separately
ECDSA	KAT: signature generation and verification are tested separately
RSA	KAT: encryption and decryption are tested separately KAT: signature generation and verification are tested separately
SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512	KAT
HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384 and HMAC-SHA-512	KAT
NIST SP800-90A Hash DRBG	KAT
Module Integrity	DSA signature verification with 2048 bit key and SHA-256

Table 12 – Power-Up Self-Tests

2.7.2 Conditional Self-Tests

The following table provides the lists of Pairwise Consistency Tests (PCT) and Continuous Random Number Generator Test (CRNGT) as the conditional self-tests. If any of the conditional test fails, the Module enters the Error sate. It returns the error code CKR_DEVICE_ERROR to the calling application to indicate the Error sate. The Module needs to be reinitialized in order to recover from the Error sate.

Test Target	Description
DSA	PCT for DSA key generation
ECDSA	PCT for ECDSA key generation
RSA	PCT for RSA key generation
NIST SP800-90A DRBG	CRNGT

Table 13 – Conditional Self-Tests

2.8 Mitigation of Other Attacks

The Module is designed to mitigate the following attacks:

Attack	Mitigation Mechanism	Specific Limit
Timing attacks on RSA	**RSA Blinding** Timing attack on RSA Was first demonstrated by Paul Kocher in 1996 [P.Kocher], who contributed the mitigation code to our module. Most recently Boneh and Brumley [D. Boneh] showed that RSA blinding is an effective defense against timing attacks on RSA.	None
Cache-timing attacks on the modular exponentiation operation used In RSA and DSA	**Cache invariant module exponentiation** This is a variant of a modular exponentiation implementation that Colin Percival [C. Percival] showed to defend against cache-timing attacks	This mechanism requires intimate knowledge of the cache line sizes of the processor. The mechanism may be ineffective when the module is running on a processor whose chase line sizes are unknown.
Arithmetic errors in RSA signatures	**Double-checking RSA signatures** Arithmetic errors in RSA signatures might leak the private key. Ferguson and Schneier [N. Ferguson] recommend that every RSA signature generation should verify the signature just generated.	None

Table 14 – Mitigation of Other Attacks

3 Security Rules and Guidance

3.1 Crypto Officer Guidance

The version of the RPMs containing the FIPS validated Module is sated in section 1.3. The RPM packages forming the Module can be installed by standard tools recommended for the installation of RPM packages in RED HAT Enterprise Linux system (for example, yum, rpm, and the RHN remote management tool). All RPM packages are signed with the Red Hat build key, which is an RSA 2048 bit key using SHA-256 signatures. The signature is automatically verified upon installation of the RPM package. If the signature cannot be validated, the RPM tool rejects the installation of the package. In such a case, the Crypto Officer is requested to obtain a new copy of the module's RPMs from Red Hat.

In addition, to support the Module, the NSPR library must be installed that is offered by the underlying operating system.
Only the cipher types listed in section 1.2 are allowed to be used.
To bring the Module into FIPS Approved mode, perform the following:

1. Install the dracut-fips package:
   ```
   # yum install dracut-fips
   ```
2. Recreate the INITRAMFS image:
   ```
   # dracut -f
   ```

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:

```
fips=1
```

If /boot or /boot/efi resides on a separate partition, the kernel parameter boot=<partition of /boot or /boot/efi> must be supplied. 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 following string needs to be appended to the kernel command line:

```
"boot=/dev/sda1"
```

Reboot to apply these settings.
If an application that uses the Module for its cryptography is put into a chroot environment, the Crypto Officer must ensure one of the above methods is available to the Module from within the chroot environment to ensure entry in FIPS Approved mode. Failure to do so will not allow the application to properly enter FIPS Approved mode.

3.1.1 Access to Audit Data

The Module may use the Unix syslog function and the audit mechanism proved by the operating system to audit events. Auditing is turned off by default. Auditing capability must be turned on as part of the initialization procedures by setting the environment variable NSS_ENABLE_AUDIT to 1. The Crypto-Officer must also configure the operating system's audit mechanism.

The Module uses the syslog function to audit events, so that audit data are stored int the system log. Only the root user can modify the system log. Ion some platforms, only the root user can read the system log: on other platforms, all user can read the system log. The system log is usually under /var/log directory. The exact location of the system log is specified in the /ext/syslog.conf file. The Module uses the default user facility and the info, warning, and err severity levels for its log messages.

The Module can also be configured to use the audit mechanism provided by the operating system to audit events. The audit data would then be stored in the system audit log. Only the root user can read or modify the system audit log. To turn on this capability it is necessary to create a symbolic link from the library file /usr/lib/libaudit.so.0 to /usr/lib.libaudit.so.1.0.0 (on 32-bit platforms) and /usr/lib64/libaudit.so.0 to /usr/lib64/libaudit.so.1.0.0 (on 64-bit platforms).

3.2 User Guidance

The Module must be operated in FIPS Approved mode to ensure that FIPS 140-2 validated cryptographic algorithms and security functions are used.
The following module initialization steps must be followed by the Crypto-Officer before starting to use the NSS module:

- Set the environment variables NSS_ENABLE_AUDIT to 1 before using the Module with an application
- Use the application to get the function pointer list using the API "FC_GetFunctionList".
- Use the API FC_Initialize to initialize the module and ensure that it returns CKR_OK. A return code other than CKR_OK means the Module is not initialized correctly, and in that case, the module must be reset and initialized again.
- For the first login, provide a NULL password and login using the fucntion pointer C_Login, which will in-turn call FC_Login API of the Module. This is required to set the initial NSS User password.
- Now, set the initial NSS User role password using the function pointer C_INITPIN. This will call the module's API FC_InitPIN API. Then, logout suing the function pointer C_Logput, which will call the module's API FC_Logout.
- The NSS User role can now be assumed on the Module by logging in using the User password. The Crypto-Officer role can be implicitly assumed by performing the Crypto=Officer services as listed in section 3.1.

The Module can be configured to use different private key database formats: key3.db or key4.db. “key3.db” format is based on the Berkeley DataBase engine and should be used by more than one process concurrently. “key4.db” format is based on SQL DataBased engine and can be used concurrently by multiple processes. Both databases are considered outside the Module’s logical boundary and all data stored in these databases is considered stored in plaintext. The interface code of the Module that accesses data stored in the database is considered part of the cryptographic boundary.
Secret and private keys. [plaintext passwords and other security-relevant data items are maintained under the control of the cryptographic module. Secret and private keys must be passed to the calling application in encrypted from with FC_UnwrapKey and entered from calling application in encrypted form with FC_UnwrapKey. The key transport methods allowed for this purpose in FIPS Approved mode are AES, Triple-DES and RSA key wrapping using the corresponding Approved modes and key sizes. Note: If the secret and private keys passed to the calling application 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.

Automated key transport methods must use FC_WrapKey and FC_UnwrapKey to output or input secret and private keys from or to the module. Note these are available in Non-Approved mode only.

All Cryptographic keys used in the FIPS Approved mode of operation must be generated in the FIPS Approved mode or imported while running in the FIPS Approved mode.

3.2.1 TLS Operations

The Module does not implement the TLS protocol. The Module implements the cryptographic operations, including TLS-specific key generation and derivation operations, which can be used to implement the TLS protocol.

3.2.2 RSA and DSA Keys

The Module allows the use of 1024 bits RSA and DSA keys for legacy purposes including signature generation, which is disallowed to be used in FIPS Approved mode as per NIST SP800-131A. Therefore, the cryptographic operations with the non-approved key sizes will result in the module operating in non-Approved mode implicitly.

The Approved algorithms shall not use the RSA keys generated by the module’s non-Approved RSA key generation method.

3.3 Handling Self-Test Errors

When the Module enters the Error state, it needs to be reinitialized to resume normal operation. Reinitialization is accomplished by calling FC_Finalize followed by FC_Initialize.

3.4 Basic Enforcement

The module design corresponds to the Module security rules. This section documents the security rules enforced by the cryptographic module to implement the security requirements of this FIPS 140-2 Level 1 module.

1. The module provides two distinct operator roles: User and Cryptographic Officer.
2. The module does not provide authentication.
3. The operator may command the module to perform the power up self-tests by cycling power or resetting the module.
4. Power-up self-tests do not require any operator action.
5. Data output is inhibited during key generation, self-tests, zeroization, and error states.
6. Status information does not contain CSPs or sensitive data that if misused could lead to a compromise of the module.
7. There are no restrictions on which keys or CSPs are zeroized by the zeroization service.
8. The module does not support concurrent operators.
9. The module does not have any external input/output devices used for entry/output of data.
10. The module does not enter or output plaintext CSPs from the module's physical boundary.
11. The module does not output intermediate key values.

4 References and Acronyms

4.1 References

Abbreviation	Full Specification Name
FIPS 140-2	*Security Requirements for Cryptographic modules, May 25, 2001*
FIPS 180-4	*Secure Hash Standard (SHS)*
FIPS 186-4	*Digital Signature Standard (DSS)*
FIPS 197	*Advanced Encryption Standard*
FIPS 198-1	*The Keyed-Hash Message Authentication Code (HMAC)*
IG	*Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program*
PKCS#1 v2.1	*RSA Cryptography Standard*
PKCS#5	*Password-Based Cryptography Standard*
PKCS#11	*"PKCS #11 v2.20: Cryptographic Token Interface Standard", 2004.*
SP 800-38A	*Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode*
SP 800-38D	*Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*
SP 800-38F	*Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*
SP 800-56A	*Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*
SP 800-67	*Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*
SP 800-89	*Recommendation for Obtaining Assurances for Digital Signature Applications*
SP 800-90A	*Recommendation for Random Number Generation Using Deterministic Random Bit Generators*
P.Kocher	*P.Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", CRYPTO '96 Lecture Notes in Computer Science, Vol. 1109, pp. 104-113, Springer Verlag, 1996. http:ll www.cryptography. com/t imingatt ack/*
D. Boneh	*D. Boneh and D. Brumley. "Remote Timing Attacks are Practical", http://crypt o.stanford.edu/- dabo/abstracts/ssl-t iming.html*
C. Percival	*C. Percival, "Cache Missing for Fun Profit",* http://www.daemonology.net/papers/htt.pdf
N. Ferguson	*N. Ferguson and B. Schneier, Practical Cryptography, Sec. 16.1.4 "Checking RSA Signatures", p. 286, Wiley Publishing, Inc., 2003.*

Table 15 – References

4.2 Acronyms

The following table defines acronyms found in this document:

Acronym	Term
AES	Advanced Encryption Standard
API	Application Programming Interface
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher-Block Chaining
CCM	Counter with CBC-MAC
CMVP	Cryptographic Module Validation Program
CO	Crypto Officer
CPU	Central Processing Unit
CSP	Critical Security Parameter
CTR	Counter-mode
CVL	Component Validation List
DES	Data Encryption Standard
DRAM	Dynamic Random Access Memory
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FCC	Federal Communications Commission
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
GMAC	Galois Message Authentication Code
GPC	General Purpose Computer
HMAC	(Keyed-) Hash Message Authentication Code
IG	Implementation Guidance
IV	Initialization Vector
KAS	Key Agreement Scheme
KAT	Known Answer Test
MAC	Message Authentication Code
MD5	Message Digest algorithm MD5
N/A	Non Applicable
NDRNG	Non Deterministic Random Number Generator
NIST	National Institute of Science and Technology
OCB	Offset Codebook Mode
OFB	Output Feedback
OS	Operating System

Acronym	Term
PKCS	Public-Key Cryptography Standards
RSA	Rivest, Shamir, and Adleman
SHA	Secure Hash Algorithm
TCBC	TDEA Cipher-Block Chaining
TCFB	TDEA Cipher Feedback Mode
TDEA	Triple Data Encryption Algorithm
TDES	Triple Data Encryption Standard
TECB	TDEA Electronic Codebook
TOFB	TDEA Output Feedback
TLS	Transport Layer Security
USB	Universal Serial Bus

Table 16 – Acronyms and Terms