Amazon Linux 2 NSS Cryptographic Module

Module Version 1.0

FIPS 140-2 Non-Proprietary Security Policy

Document Version 1.2

Last update: 2020-Mar-30

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

© 2020 Amazon Web Services, Inc./atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Table of Contents

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

List of Tables

List of Figures

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Copyrights and Trademarks

Amazon is a registered trademark of Amazon Web Services, Inc. or its affiliates.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

1 Introduction

This document is the non-proprietary FIPS 140-2 Security Policy for version 1.0 of the Amazon Linux 2 NSS Cryptographic Module. It contains the security rules under which the module must be operated and describes how this module meets the requirements as specified in FIPS 140-2 (Federal Information Processing Standards Publication 140-2) for a Security Level 1 module.

1.1 Purpose of the Security Policy

There are three major reasons that a security policy is needed:

- It is required for FIPS 140 2 validation,
- It allows individuals and organizations to determine whether a cryptographic module, as implemented, satisfies the stated security policy, and
- It describes the capabilities, protection and access rights provided by the cryptographic module, allowing individuals and organizations to determine whether it will meet their security requirements.

1.2 Target Audience

This document is part of the package of documents that are submitted for FIPS 140 2 conformance validation of the module. It is intended for the following audience:

- Developers.
- FIPS 140-2 testing lab.
- The Cryptographic Module Validation Program (CMVP).
- Customers using or considering integration of Amazon Linux 2 NSS Cryptographic Module.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

2 Cryptographic Module Specification

2.1 Module Overview

The Amazon Linux 2 NSS Cryptographic Module (hereafter referred to as the “module”) is a set of libraries designed to support cross-platform development of security-enabled applications. These applications may support the TLS protocol, PKCS #5, PKCS #7, PKCS #11, PKCS #12, S/MIME, X.509 v3 certificates, and other security standards supporting FIPS 140-2 validated cryptographic algorithms. The module provides a C language Application Program Interface (API) for use by other calling applications that require cryptographic functionality.

The module provides support for AESNI instruction set from the CPU for AES and C-language generic implementations for the algorithms. Although the module supports more than one implementation of algorithms, only one implementation of an algorithm will be available at runtime.

2.2 FIPS 140-2 Validation Scope

Table 1 shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2 standard.

Table 1: FIPS 140-2 Security Requirements.

Security Requirements Section		Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles and Services and Authentication	2
4	Finite State Machine Model	1
5	Physical Security	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self-Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	1
Overall Level		1

2.3 Definition of the Cryptographic Module

The Amazon Linux 2 NSS Cryptographic Module is defined as a Software, Multi-chip Standalone module per the requirements within FIPS 140-2. The logical cryptographic boundary of the module consists of shared library files and their integrity test HMAC files, which are delivered through the Amazon Linux 2 yum core repository (ID amz2-core/2/x86_64) from the following RPM files:

- nss-softokn-3.36.0-5.amzn2.0.1.x86_64.rpm
- nss-softokn-freebl-3.36.0-5.amzn2.0.1.x86_64.rpm

Table 2 summarizes the components of the cryptographic module.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Table 2: Components of the module.

Component	Description
/usr/lib64/libnssdbm3.so	This library provides database storage implementations.
/usr/lib64/libsoftokn3.so	This library provides exposes FreeBL functionality through a PKCS#11 interface.
/usr/lib64/libfreeblpriv3.so	This FreeBL library includes implementations for big number computations and cryptographic algorithms.
/usr/lib64/libnssdbm3.chk	Signature (DSA with 2048-bit key and SHA-256) value of the associated library for integrity check during the power-on.
/usr/lib64/libsoftokn3.chk	Signature (DSA with 2048-bit key and SHA-256) value of the associated library for integrity check during the power-on.
/usr/lib64/libfreeblpriv3.chk	Signature (DSA with 2048-bit key and SHA-256) value of the associated library for integrity check during the power-on.

Figure 1 shows the logical block diagram of the module executing in memory on the host system. The logical cryptographic boundary is indicated with a dashed colored box.

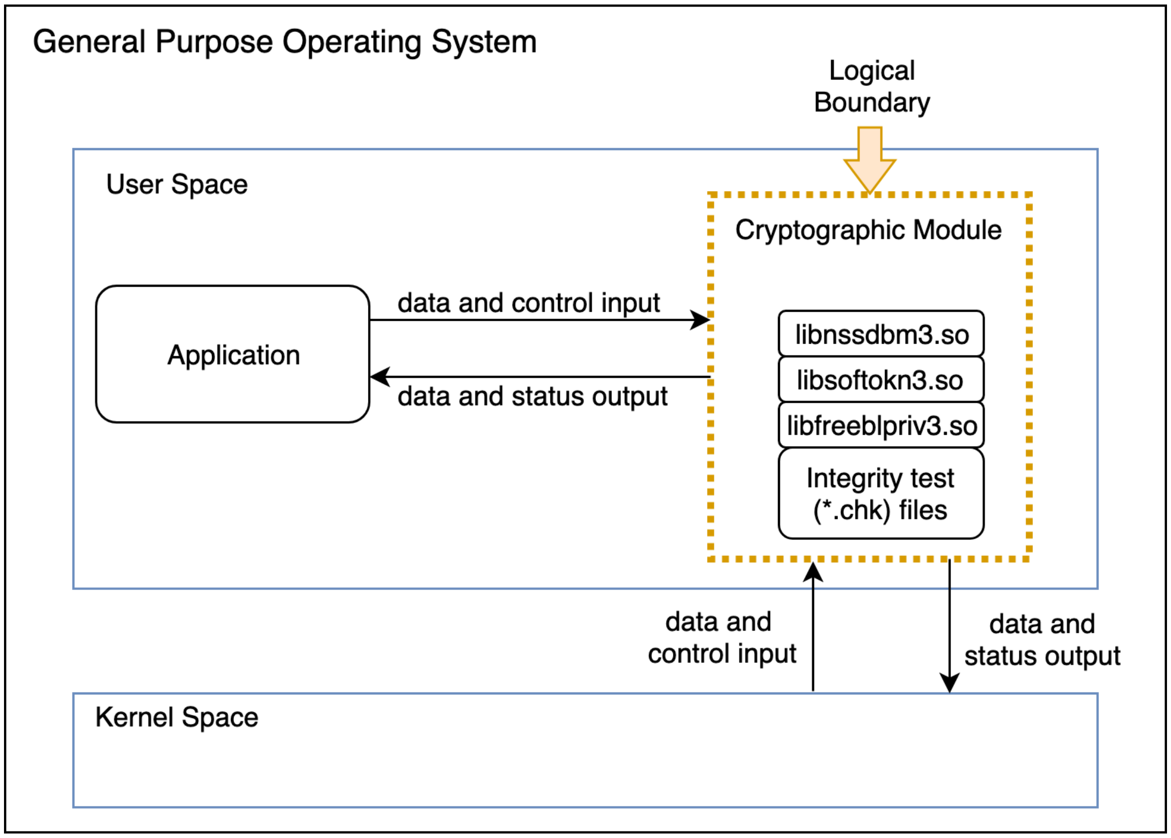

Figure 1: Logical cryptographic boundary.

2.4 Definition of the Physical Cryptographic Boundary

The physical cryptographic boundary of the module is defined as the hard enclosure of the host system on which the module runs. Figure 2 depicts the hardware block diagram. The physical hard enclosure is indicated by the dashed colored line. No components are excluded from the requirements of FIPS 140-2.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

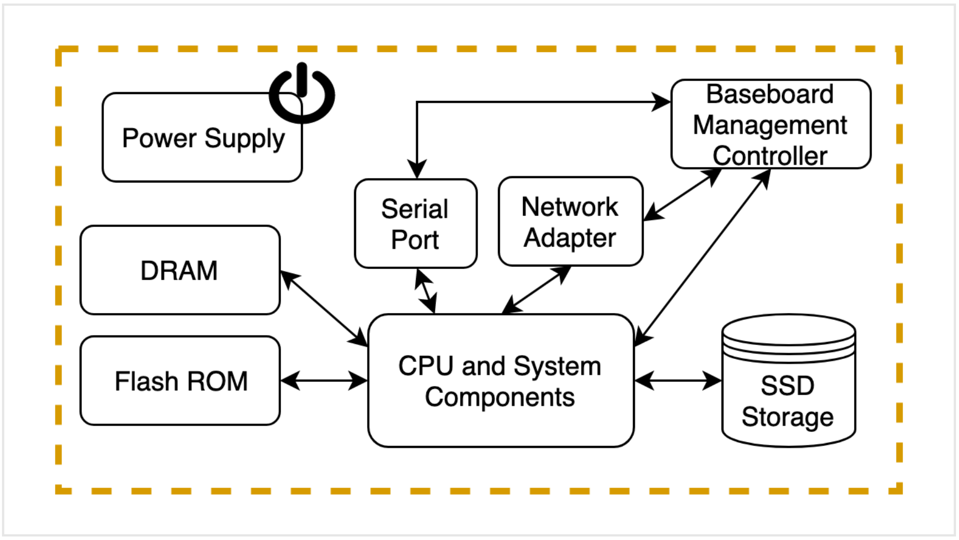

Figure 2: Hardware block diagram.

2.5 Tested Operational Environments

The module was tested on the environments/platforms listed in Table 3. The tested operational environment was controlled and the laboratory had full and exclusive access to the environment and module during the testing procedures.

Table 3: Tested operational environments.

Operating System	Processor	Hardware
Amazon Linux 2	Intel ® Xeon ® E5 (Broadwell) x86_64bit with PAA (i.e., AES-NI)	Amazon EC2 i3.metal 512 GiB system memory 13.6 TiB SSD storage + 8 GiB SSD boot disk 25 Gbps Elastic Network Adapter
Amazon Linux 2	Intel ® Xeon ® E5 (Broadwell) x86_64bit without PAA (i.e., AES-NI)	Amazon EC2 i3.metal 512 GiB system memory 13.6 TiB SSD storage + 8 GiB SSD boot disk 25 Gbps Elastic Network Adapter

2.6 Modes of Operation

The module supports two modes of operation.

- In "**FIPS mode**" (the Approved mode of operation), only approved or allowed security functions with sufficient security strength are offered by the module.
- In "**non-FIPS mode**" (the non-Approved mode of operation), non-approved security functions are offered by the module.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

The module enters the operational mode after Power-On Self-Tests (POST) succeed. Once the module is operational, the mode of operation is implicitly assumed depending on the security function invoked and the security strength[1] of the cryptographic keys chosen for the service.

If the POST or the Conditional Tests fail (Section 9), the module goes into the error state. The status of the module can be determined by the availability of the module. If the module is available, then it had passed all self-tests. If the module is unavailable, it is because any self-test failed, and the module has transitioned to the error state.

Keys and Critical Security Parameters (CSPs) used or stored in FIPS mode shall not be used in non-FIPS mode, and vice versa.

[1] See Section 5.6.1 in [SP800-57] for a definition of “security strength”.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

3 Module Ports and Interfaces

As a Software module, the module does not have physical ports. The operator can only interact with the module through the API provided by the module. Thus, the physical ports within the physical boundary are interpreted to be the physical ports of the hardware platform on which the module runs and are directed through the logical interfaces provided by the software.

The logical interfaces are the API through which applications request services and receive output data through return values or modified data referenced by pointers; database files in the file system, and environment variables. The module distinguishes between data input, control input, data output and status output by using different function arguments for each of those paths. In this manner, the logical paths followed by data and control entering the module and data and status exiting the module are not connected. The module does not use the same buffer for input and output. After the input buffer that holds security-related data is no longer used, the module zeroizes the memory area occupied by the buffer, thus preventing leakage of sensitive information in case the same memory area is later utilized.

Table 4 summarizes the logical interfaces and the power input.

Table 4: Ports and interfaces.

Logical Interface	Description
Data Input	API input parameters and database files in file system.
Data Output	API output parameters and database files in file system.
Control Input	API function calls and environment variables.
Status Output	API return codes and status parameters.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

4 Roles, Services and Authentication

4.1 Roles

The module supports the following roles:

- **User role**: performs all services (in both FIPS mode and non-FIPS mode of operation), except module installation and configuration. This role is assumed by the calling application accessing the module. The user role is also responsible for the retrieval, updating and deletion of keys from the private key database.
- **Crypto Officer role**: performs module installation and configuration. In addition, the Crypto Office can access status services and other services that do not utilize secret/private keys and other CSPs associated with the User, such as message digest and random number generation. The Crypto Office is responsible for access control to the module before and after installation, including management of physical access to the general-purpose computer, execution of the module code and management of the security facilities provided by the operating system.

4.2 Role Assumption and Operator Authentication

The module implements role-based authentication.

The Crypto Officer role is implicitly assumed by an operator while installing the module by following the instructions in Section 10.2 and while performing any other services available to the Crypto Officer on the module.

For the User role, the module implements a password-based authentication mechanism. 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 this password is not displayed. The return value of the function is the only feedback from the authentication mechanism, and does not contain any information that could be used to guess or determine the User's password. The password is initialized by the Crypto Officer role as part of the module initialization process, 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 the module is powered off, the result of the previous authentication is cleared. The user needs to be re-authenticated in a subsequent power-on.

4.2.1 Strength of the Operator Authentication Mechanism

The 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 extracted from three or more character classes. The character classes are:
  1. Digits (0-9).
  2. ASCII lowercase letters (a-z).
  3. ASCII uppercase letters (A-Z).
  4. ASCII non-alphanumeric characters (space and other ASCII special characters such as '$', '!').

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

5. 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, two assumptions are considered.

- The characters of the password are independent with each other.
- The password contains the smallest combination of the character classes, namely five digits, one ASCII lowercase letter and one ASCII uppercase letter.

The probability of guessing one digit is 1/10. The probability of guessing 5 digits is thus (1/10)ˆ5. The probability of guessing one lowercase letter is the same as guessing one uppercase letter, which is 1/26 in the ASCII alphabet.

In the aforementioned configuration, the probability of guessing every character successfully in the 7-character password (i.e., the probability that a random guess of the password will succeed) is (1/10)ˆ5 * (1/26) * (1/26) = 1/67,600,000. . This probability is less than the required threshold of 1/1,000,000 chance that a random attempt will succeed in obtaining authentication.

After each failed authentication attempt, the NSS cryptographic module inserts a one-second delay before returning the control to the caller, and the module allows 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). This probability figure is smaller than the required probability threshold of 1/100,000 chance that, for multiple attempts to use the authentication mechanism during a one-minute period, a random attempt will success in obtaining authentication.

4.3 Services

The module supports services for the Crypto Officer and User roles. The Crypto Officer role requires no operator authentication, whereas the User role requires operator authentication. Crypto Officer services do not access secret/private keys or other CSPs associated with the User role.

4.3.1 Calling Convention of API Functions

The module has a set of API functions denoted by FC_xxx. The services offered by the module are associated with one or more of these API functions.

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. FC_GetFunctionList returns 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.

For instance, to call the FC_Initialize function that initializes the module library, first a call to FC_GetFunctionList function is performed. The CK_FUNCTION_LIST structure is returned, containing C_Initialize pointer. This C_Initialize pointer will then point to the desired FC_Initialize function. This convention is utilized in Table 5 and Table 6 wherein function names are listed.

The service tables next indicate the service name, the associated function returned by C_Initialize pointer, a description of the service, keys and CSPs touched by the service as applicable, and the type of access to these keys and CSPs. The service tables use the following convention when specifying the access permissions that the module has for each CSP or key.

- **Create (C)**: the calling application can create a new CSP.
- **Read (R)**: the calling application can read the CSP.
- **Update (U)**: the calling application can write a new value to the CSP.
- **Zeroize (Z)**: the calling application can zeroize the CSP.

- **N/A**: the calling application does not access any CSP or key during its operation.

For the “Role” column, U indicates the User role, and CO indicates the Crypto Officer role. A checkmark symbol marks which role has access to that service.

4.3.2 Services in the FIPS-Approved Mode of Operation

Table 5 provides a full description of FIPS Approved services and the non-Approved but Allowed services provided by the module in the FIPS-approved mode of operation and lists the roles allowed to invoke each service. If the service does not require authentication, i.e., if the service is non-authenticated, this characteristic will be indicated in the service name. For services in which some functions are authenticated and other are not, the functions that are not authenticated will be indicated in the Description column.

Note: the module does not implement the TLS protocol, but rather the cryptographic algorithms (such as the TLS KDF) that can be used to implement the TLS protocol by user applications.

Table 5: Services in the FIPS-approved mode of operation.

Service Name	Function	Service Description	Role		Keys and CSPs	Access
			U	CO		
Get Function List (non-authenticated)	FC_GetFunctionList	Return a pointer to the list of function pointers for the operational mode.	✓	✓	None	N/A
Module Initialization (non-authenticated)	FC_InitToken	Initialize or re-initialize a token.	✓	✓	User password, any key type	Z
	FC_InitPIN	Initialize the user's password, i.e., set the user's initial password		✓	User password	C, R, U
General Purpose (non-authenticated)	FC_Initialize	Initialize the module library	✓	✓	None	N/A
	FC_Finalize	Finalize (shut down) the module library	✓	✓	Any key type	Z
	FC_GetInfo	Obtain general information about the module library	✓	✓	None	N/A
Slot and Token Management	FC_GetSlotList	Obtain a list of slots in the system. (non-authenticated)	✓	✓	None	N/A
	FC_GetSlotInfo	Obtain information about a particular slot. (non-authenticated)	✓	✓	None	N/A
	FC_GetTokenInfo	Obtain information about the token (this function provides the Show Status service). (non-authenticated)	✓	✓	None	N/A
	FC_GetMechanismList	Obtain a list of mechanisms (cryptographic algorithms) supported by a token. (non-authenticated)	✓	✓	None	N/A
	FC_GetMechanismInfo	Obtain information about a particular mechanism. (non-authenticated)	✓	✓	None	N/A
	FC_SetPIN	Change the user's password	✓		User password	R, U

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Service Name	Function	Service Description	Role		Keys and CSPs	Access
			U	CO		
Session Management (non-authenticated)	FC_OpenSession	Open a connection (session) between an application and a particular token	✓	✓	None	N/A
	FC_CloseSession	Close a session	✓	✓	Any key type for the session	Z
	FC_CloseAllSessions	Close all sessions with a token	✓	✓	Any key type	Z
	FC_GetSessionInfo	Obtain information about the session (this function provides the Show Status service)	✓	✓	None	N/A
	FC_GetOperationState	Save the state of the cryptographic operations in a session (this function is only implemented for message digest operations)	✓	✓	None	N/A
	FC_SetOperationState	Restore the state of the cryptographic operations in a session (this function is only implemented for message digest operations)	✓	✓	None	N/A
	FC_Login	Log into a token	✓	✓	User Password	C, R, U
	FC_Logout	Log out from a token	✓	✓	None	N/A
Object Management	FC_CreateObject	Create a new object	✓		Any key type	C, U
	FC_CopyObject	Create a copy of an object	✓		Any key type	C, R, U
	FC_DestroyObject	Destroy an object	✓		Any key type	Z
	FC_GetObjectSize	Obtain the size of an object in bytes	✓		Any key type	R
	FC_GetAttributeValue	Obtain an attribute value of an object	✓		Any key type	R
	FC_SetAttributeValue	Modify an attribute value of an object	✓		Any key type	R, U
	FC_FindObjectsInit	Initialize an object search operation	✓		None	N/A
	FC_FindObjects	Continue an object search operation	✓		Any key type matching the search criteria	R
	FC_FindObjectsFinal	Finish an object search operation	✓		None	N/A
Encryption and Decryption	FC_EncryptInit	Initialize an encryption operation	✓		AES/Triple-DES key	R
	FC_Encrypt	Encrypt single-part data	✓		AES/Triple-DES key	R
	FC_EncryptUpdate	Continue a multiple-part encryption operation	✓		AES/Triple-DES key	R
	FC_EncryptFinal	Finish a multiple-part encryption operation	✓		AES/Triple-DES key	R
	FC_DecryptInit	Initialize a decryption operation	✓		AES/Triple-DES key	R

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Service Name	Function	Service Description	Role		Keys and CSPs	Access
			U	CO		
	FC_Decrypt	Decrypt single-part encrypted data	✓		AES/Triple-DES key	R
	FC_DecryptUpdate	Continue a multiple-part decryption operation	✓		AES/Triple-DES key	R
	FC_DecryptFinal	Finish a multiple-part decryption operation	✓		AES/Triple-DES key	R
Message Digest	FC_DigestInit	Initialize a message digesting operation. (non-authenticated)	✓	✓	None	N/A
	FC_Digest	Digest single-part data. (non-authenticated)	✓	✓	None	N/A
	FC_DigestUpdate	Continue a multiple-part digesting operation. (non-authenticated)	✓	✓	None	N/A
	FC_DigestKey	Continue a multiple-part message-digesting operation by digesting the value of a secret key as part of the data already digested	✓		HMAC key	R
	FC_DigestFinal	Finish a multiple-part digesting operation. (non-authenticated)	✓	✓	None	N/A
Signature Generation and Verification	FC_SignInit	Initialize a signature operation	✓		DSA/ECDSA/RSA private key, HMAC key	R
	FC_Sign	Sign single-part data	✓		DSA/ECDSA/RSA private key, HMAC key	R
	FC_SignUpdate	Continue a multiple-part signature operation	✓		DSA/ECDSA/RSA private key, HMAC key	R
	FC_SignFinal	Finish a multiple-part signature operation	✓		DSA/ECDSA/RSA private key, HMAC key	R
	FC_SignRecoverInit	Initialize a signature operation, wherein the data can be recovered from the signature	✓		DSA/ECDSA/RSA private key	R
	FC_SignRecover	Sign single-part data, wherein the data can be recovered from the signature	✓		DSA/ECDSA/RSA private key	R
	FC_VerifyInit	Initialize a verification operation	✓		DSA/ECDSA/RSA public key, HMAC key	R
	FC_Verify	Verify a signature on single-part data	✓		DSA/ECDSA/RSA public key, HMAC key	R

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Service Name	Function	Service Description	Role		Keys and CSPs	Access
			U	CO		
	FC_VerifyUpdate	Continue a multiple-part verification operation	✓		DSA/ECDSA/RSA public key, HMAC key	R
	FC_VerifyFinal	Finish a multiple-part verification operation	✓		DSA/ECDSA/RSA public key, HMAC key	R
	FC_VerifyRecoverInit	Initialize a verification operation, wherein the data is recovered from the signature	✓		DSA/ECDSA/RSA public key	R
	FC_VerifyRecover	Verify a signature on single-part data, wherein the data is recovered from the signature	✓		DSA/ECDSA/RSA public key	R
Dual Function Cryptographic Operations	FC_DigestEncryptUpdate	Continue a multiple-part digesting and encryption operation	✓		AES/Triple-DES key	R
	FC_DecryptDigestUpdate	Continue a multiple-part decryption and digesting operation	✓		AES/Triple-DES key	R
	FC_SignEncryptUpdate	Continue a multiple-part signing and encryption operation	✓		DSA/ECDSA/RSA private key, HMAC key, AES/Triple-DES key	R
	FC_DecryptVerifyUpdate	Continue a multiple-part decryption and verify operation	✓		DSA/ECDSA/RSA public key, HMAC key, AES/Triple-DES key	R
Key Management	FC_GenerateKey	Generate a secret key (also used by TLS implementations to generate a pre-master secret)	✓		AES/Triple-DES/HMAC key, TLS pre-master secret	C, U
	FC_GenerateKeyPair	Generate a public/private key pair (this function performs the pair-wise consistency tests)	✓		DSA/ECDSA/RSA key pair, Diffie-Hellman/EC Diffie-Hellman key pair	C, U
	FC_WrapKey	Wrap (encrypt) a key using SP800-38F AES key wrapping, RSA encryption	✓		AES key, RSA public key, wrapped key (of any key type)	R
	FC_UnwrapKey	Unwrap (decrypt) a key using SP800-38F AES key unwrapping, RSA decryption	✓		AES key, RSA private key, wrapped key (of any key type)	R, U
	FC_DeriveKey	Compute the shared secret (TLS pre-master secret) Derive the TLS master secret (from the TLS master secret) Derive a key from TLS master secret	✓		TLS pre-master secret TLS master secret Derived key (AES, Triple-DES, HMAC)	C, R, U

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Service Name	Function	Service Description	Role		Keys and CSPs	Access
			U	CO		
					TLS KDF internal state	
Random Number Generation (non-authenticated)	FC_SeedRandom	Mix in additional seed material to the random number generator	✓	✓	Entropy input string, seed, DRBG V and C values	R, U
	FC_GenerateRandom	Generate random data (this function performs the continuous random number generator test)	✓	✓	Random data, DRBG V and C values	R, U
Parallel Function Management	FC_GetFunctionStatus	A legacy function, which simply returns the value 0x00000051 (function not parallel)	✓		None	N/A
	FC_CancelFunction	A legacy function, which simply returns the value 0x00000051 (function not parallel)	✓		None	N/A
Self-Tests (non-authenticated)	N/A	Self-tests for all cryptographic functions, performed automatically when loading the module	✓	✓	DSA 2048-bit public key for module integrity test	R
Zeroization	FC_DestroyObject FC_InitToken FC_Finalize FC_CloseSession FC_CloseAllSessions	Zeroization of keys and CSPs; all CSPs are automatically zeroized when freeing the cipher handle	✓		Any keys, CSPs	Z
Module Installation (non-authenticated)	N/A	Installation of the module		✓	None	N/A
Module Configuration (non-authenticated)	N/A	Configuration of the module		✓	None	N/A

4.3.3 Services in the Non-FIPS-Approved Mode of Operation

Table 6 presents the services only available in non-FIPS-approved mode of operation, as these services invoke algorithms, or use keys and elliptic curves not listed in Table 7. The associated functions are the same as the ones listed in Table 5, however the service names specify the conditions under which those services are made available only in the non-FIPS-approved mode of operation. If the service does not require authentication, i.e., if the service is non-authenticated, this characteristic will be indicated in the service name. For services in which some functions are authenticated and other are not, the functions that are not authenticated will be indicated in the Description column.

Invoking any of the services in Table 6 will implicitly switch the module to the non-FIPS-approved mode.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Table 6: Services in the non-FIPS approved mode of operation.

Service Name	Function	Service Description	Role		Keys	Access
			U	CO		
Encryption and Decryption using symmetric algorithms not listed in Table 7 (e.g., Camellia, DES, RC2, RC4, etc. See Table 9)	FC_EncryptInit	Initialize an encryption operation	✓		Symmetric key	R
	FC_Encrypt	Encrypt single-part data	✓		Symmetric key	R
	FC_EncryptUpdate	Continue a multiple-part encryption operation	✓		Symmetric key	R
	FC_EncryptFinal	Finish a multiple-part encryption operation	✓		Symmetric key	R
	FC_DecryptInit	Initialize a decryption operation	✓		Symmetric key	R
	FC_Decrypt	Decrypt single-part encrypted data	✓		Symmetric key	R
	FC_DecryptUpdate	Continue a multiple-part decryption operation	✓		Symmetric key	R
	FC_DecryptFinal	Finish a multiple-part decryption operation	✓		Symmetric key	R
Message Digest with MD2, MD5	FC_DigestInit	Initialize a message digesting operation. (non-authenticated)	✓	✓	None	N/A
	FC_Digest	Digest single-part data. (non-authenticated)	✓	✓	None	N/A
	FC_DigestUpdate	Continue a multiple-part digesting operation. (non-authenticated)	✓	✓	None	N/A
	FC_DigestKey	Continue a multiple-part message-digesting operation by digesting the value of a secret key as part of the data already digested	✓		MAC Key	R
	FC_DigestFinal	Finish a multiple-part digesting operation. (non-authenticated)	✓	✓	None	N/A
Signature Generation and Verification with RSA, DSA, ECDSA key sizes and elliptic curves not listed in Table 7; signature generation with SHA-1	FC_SignInit	Initialize a signature operation	✓		DSA/ECDSA/RSA private key, HMAC key	R
	FC_Sign	Sign single-part data	✓		DSA/ECDSA/RSA private key, HMAC key	R
	FC_SignUpdate	Continue a multiple-part signature operation	✓		DSA/ECDSA/RSA private key, HMAC key	R
	FC_SignFinal	Finish a multiple-part signature operation	✓		DSA/ECDSA/RSA private key, HMAC key	R
	FC_SignRecoverInit	Initialize a signature operation, wherein the data can be recovered from the signature	✓		DSA/ECDSA/RSA private key	R

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Service Name	Function	Service Description	Role		Keys	Access
			U	CO		
	FC_SignRecover	Sign single-part data, wherein the data can be recovered from the signature	✓		DSA/ECDSA/RSA private key	R
	FC_VerifyInit	Initialize a verification operation	✓		DSA/ECDSA/RSA public key, HMAC key	R
	FC_Verify	Verify a signature on single-part data	✓		DSA/ECDSA/RSA public key, HMAC key	R
	FC_VerifyUpdate	Continue a multiple-part verification operation	✓		DSA/ECDSA/RSA public key, HMAC key	R
	FC_VerifyFinal	Finish a multiple-part verification operation	✓		DSA/ECDSA/RSA public key, HMAC key	R
	FC_VerifyRecoverInit	Initialize a verification operation, wherein the data is recovered from the signature	✓		DSA/ECDSA/RSA public key	R
	FC_VerifyRecover	Verify a signature on single-part data, wherein the data is recovered from the signature	✓		DSA/ECDSA/RSA public key	R
Dual Function Cryptographic Operations using any algorithm, key sizes or elliptic curves not listed in Table 7	FC_DigestEncryptUpdate	Continue a multiple-part digesting and encryption operation	✓		Symmetric key	R
	FC_DecryptDigestUpdate	Continue a multiple-part decryption and digesting operation	✓		Symmetric key	R
	FC_SignEncryptUpdate	Continue a multiple-part signing and encryption operation	✓		DSA/ECDSA/RSA private key, MAC key, ymmetric key	R
	FC_DecryptVerifyUpdate	Continue a multiple-part decryption and verify operation	✓		DSA/ECDSA/RSA public key, MAC key, symmetric key	R
Key Management using any algorithm, key sizes or elliptic curves not listed in Table 7(including J-PAKE); key generation not compliant with FIPS 186-4	FC_GenerateKey	Generate a secret key (also used by TLS implementations to generate a pre-master secret)	✓		Symmetric key or key material	C, U
	FC_GenerateKeyPair	Generate a public/private key pair (this function performs the pair-wise consistency tests)	✓		aymmetric key pair	C, U
	FC_WrapKey	Wrap (encrypt) a key using SP800-38F AES key wrapping, RSA encryption	✓		Symmetric, asymmetric key (key encryption key), wrapped key (of any key type)	R
	FC_UnwrapKey	Unwrap (decrypt) a key using SP800-38F AES key unwrapping, RSA decryption	✓		Symmetric, asymmetric key (key encryption key), wrapped	R, U

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Service Name	Function	Service Description	Role		Keys	Access
			U	CO		
					key (of any key type)	
	FC_DeriveKey	Compute the shared secret (TLS pre-master secret) Derive the TLS master secret (from the TLS master secret) Derive a key from TLS master secret	✓		TLS pre-master secret, TLS master secret, derived key (AES, Triple-DES, HMAC, other algorithms not listed in Table 7)	C, R, U

4.4 Algorithms

The module implements cryptographic algorithms that are used by the services provided by the module. The cryptographic algorithms that are approved to be used in the FIPS mode of operation are tested and validated by the CAVP. No parts of the Transport Layer Security (TLS) protocol have been tested by the CAVP, but for the key derivation function (KDF).

The module supports different AES implementations based on the underlying platform's capability (per the tested operational environment in Table 3). The module supports the use of AES-NI from the Intel architecture. When the AES-NI is enabled in the operating environment, the module performs the AES operations supported by the AES-NI instructions. When the AES-NI is disabled in the operating environment, the module performs the AES operations using the C implementation.

The AES implementations that use the AES-NI instructions and their related algorithms have been tested by CAVS and subjected to functional testing. Although the module offers different implementations for AES, only one implementation for the respective algorithm will ever be available for AES cryptographic services at run-time.

Table 7, Table 8 and Table 9 present the cryptographic algorithms in specific modes of operation. These tables include the CAVP certificates for different implementations, the algorithm name, respective standards, the available modes and key sizes wherein applicable, and usage. Information from certain columns may be applicable to more than one row.

Not all algorithms present in the listed certificates are available in the FIPS-approved mode of operation of this module.

4.4.1 FIPS-Approved Algorithms

Table 7 lists the cryptographic algorithms that are approved to be used in the FIPS mode of operation.

Table 7: FIPS-approved cryptographic algorithms.

Algorithm	Standard	Mode/Method	Key size	Use	CAVP Cert#
AES	[FIPS197] [SP800-38A]	CBC, ECB, CTR	128, 192 and 256 bits	Data Encryption and Decryption	#C803 (C) #C804 (AES-NI)
	[FIPS197] [SP800-38F]	KW	128, 192, 256 bits	Key Wrapping	

Algorithm	Standard	Mode/Method	Key size	Use	CAVP Cert#
DSA	[FIPS 186-4]		L=2048, N=224; L=2048, N=256; L=3072, N=256	Key Pair Generation	#C803 (C)
		SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	L=1024, N=160	Domain Parameter Verification	
		SHA-224, SHA-256, SHA-384, SHA-512	L=2048, N=224		
		SHA-256, SHA-384, SHA-512	L=2048, N=256; L=3072, N=256		
		SHA-224, SHA-256, SHA-384, SHA-512	L=2048, N=224; L=2048, N=256; L=3072, N=256	Signature Generation	
		SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	L=1024, N=160, L=2048, N=224; L=2048, N=256; L=3072, N=256	Signature Verification	
DRBG	[SP800-90A]	HASH_DRBG with SHA-256, without Prediction Resistance	n/a	Random Number Generation	#C803 (C)
ECDSA	[FIPS186-4]	Extra bits	P-256, P-384, P-521	Key Pair Generation	#C803 (C)
			P-256, P-384, P-521	Public Key Verification	
		SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Signature Generation	
		SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Signature Verification	
KAS ECC Component	[SP800-56A]	ECC Ephemeral Unified scheme	P-256 (EC), P-384 (ED), P-521 (EE)	EC Diffie-Hellman Shared Secret Computation	#C803 (C)

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Algorithm	Standard	Mode/Method	Key size	Use	CAVP Cert#
KAS FFC Component	[SP800-56A]	FFC dhEphem scheme	p=2048, q=224 (FB); p=2048, q=256 (FC)	Diffie-Hellman Shared Secret Computation	#C803 (C)
HMAC	[FIPS198-1]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 bits or greater	Message Authentication Code	#C803 (C)
KDF Component in TLS v1.0/1.1 TLS v1.2	[SP800-135]	SHA-256, SHA-384, SHA-512		Key Derivation	#C803 (C)
RSA	[FIPS186-4]	X9.31	2048 and 3072 bits	Key Pair Generation	#C803 (C)
		PKCS#1v1.5 with SHA-224, SHA-256, SHA-384, SHA-512	2048 and 3072 bits	Digital Signature Generation	
		PKCS#1v1.5 with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024, 2048, and 3072 bits	Signature Verification	
	[FIPS186-2]	PKCS#1v1.5 with SHA-224, SHA-256, SHA-384, SHA-512	4096	Digital Signature Generation	
SHS	[FIPS180-4]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest	#C803 (C)
Triple-DES	[SP800-67] [SP800-38A]	CBC, ECB, CTR	192 bits	Data Encryption and Decryption	#C803 (C)
KTS	[SP800-38F]	AES-KW	128, 192, and 256 bits	Key Wrapping	#C803 (C) #C804 (AES-NI)
CKG	IG D.12 [SP800-133]		Defined by caller	Key Generation	Vendor Affirmed

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

4.4.2 Non-Approved-but-Allowed Algorithms

Table 8 lists the non-Approved-but-Allowed cryptographic algorithms provided by the module that are allowed to be used in the FIPS mode of operation.

Table 8: Non-Approved-but-allowed cryptographic algorithms.

Algorithm	Usage
RSA Key Wrapping with key size between 2048 bits and 15360 bits (or more)	Key wrapping, key establishment methodology provides between 112 and 256 bits of encryption strength.
Diffie-Hellman with key size between 2048 bits and 15360 bits (or more)	Shared secret computation provides between 112 and 256 bits of encryption strength. Allowed by IG A.14.
NDRNG	Used for seeding NIST SP 800-90A DRBG.
MD5	Message digest used in TLS only.

4.4.3 Non-Approved Algorithms

Table 9 lists the cryptographic algorithms that are not allowed to be used in the FIPS mode of operation. Use of any of these algorithms (and corresponding services in Table 6) will implicitly switch the module to the non-Approved mode.

Table 9: Non-FIPS approved cryptographic algorithms.

Algorithm	Usage
AES-CTS	Encryption/decryption
AES-GCM	Encryption/decryption, non-compliant with IG A.5. Tested with CAVP Certs. #C803, #C804
Camellia	Encryption/decryption
DES	Encryption/decryption
Diffie-Hellman	Shared secret computation using keys of length not listed in Table 7
DSA	Parameter/Key generation/Signature generation with keys not listed in Table 7
EC Diffie-Hellman	Shared secret computation using curves not listed in Table 7
ECDSA	Key generation/Signature generation with curves not listed in Table 7
J-PAKE	Key agreement
Key Wrapping	Key wrapping methods not compliant with SP800-38F
MD2	Hash function
MD5	Hash function
RC2	Encryption/decryption

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Algorithm	Usage
RC4	Encryption/decryption
RC5	Encryption/decryption
RSA	Key generation/Signature generation/Signature verification with keys of length not listed in Table 7
SEED	Encryption/decryption
SHA-1	Signature generation
Two-key Triple-DES	Encryption/decryption, key wrapping

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

5 Physical Security

The module is comprised of software only and thus this Security Policy does not claim any physical security.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

6 Operational Environment

6.1 Applicability

The module operates in a modifiable operational environment per FIPS 140-2 Security Level 1 specifications. The module runs on the Amazon Linux 2 operating system executing on the hardware specified in Section 2.5.

6.2 Policy

The operating system is restricted to a single operator mode of operation (i.e., concurrent operators are explicitly excluded by the operating system).

The application that makes calls to the modules is the single user of the modules, even when the application is serving multiple clients.

In operational mode, the ptrace(2) system call, the gdb(1) debugger and strace(1) shall not be used. In addition, other tracing mechanisms offered by the Linux environment, such as ftrace or systemtap, shall not be used.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

7 Cryptographic Key Management

Table 10 summarizes the public keys, secret/private keys and other CSPs that are used by the cryptographic services implemented in the module. The table describes the use of each key/CSP and, as applicable, how they are generated or established, their method of entry and output of the module, their storage location, and the method for zeroizing the key/CSP.

All key and CSP storage is done in plaintext.

Table 10: Lifecycle of public keys, secret/private keys and other Critical Security Parameters (CSPs).

Name	Use	Generation/ Establishment	Entry/ Output	Type	Storage	Zeroization
AES Key	Encryption, decryption.	Generated by SP800-90A DRBG.	Either in plaintext or Encrypted through key wrapping using FC_WrapKey.	AES key, all modes and lengths per Table 7	RAM or key database	Automatically zeroized when freeing the cipher handle
Triple-DES Keys	Encryption, decryption.	Generated by SP800-90A DRBG.	Either in plaintext or Encrypted through key wrapping using FC_WrapKey.	Triple-DES key, all modes and lengths per Table 7	RAM or key database	Automatically zeroized when freeing the cipher handle
HMAC Key	MAC generation and verification	Generated by SP800-90A DRBG.	Either in plaintext or Encrypted through key wrapping using FC_WrapKey.	HMAC keys of length > 112 bits	RAM or key database	Automatically zeroized when freeing the cipher handle
RSA public and private key	RSA signature generation and verification. Key wrapping.	Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800-90A DRBG.	Either in plaintext or Encrypted through key wrapping using FC_WrapKey.	RSA keys of length 1024 (sigver), 2048, 3072 bits (or more as allowed for key wrapping)	RAM or key database	Automatically zeroized when freeing the cipher handle
DSA public and private key	DSA signature generation and verification.	Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800-90A DRBG.	Either in plaintext or Encrypted through key wrapping using	DSA keys of length 1024 (sigver), 2048, 3072 bits	RAM or key database	Automatically zeroized when freeing the cipher handle

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Name	Use	Generation/ Establishment	Entry/ Output	Type	Storage	Zeroization
			FC_WrapKey.			
ECDSA public and private key	ECDSA signature generation and verification.	Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800-90A DRBG.	Either in plaintext or Encrypted through key wrapping using FC_WrapKey.	ECDSA keys for curves P-256, P-384, P-521	RAM or key database	Automatically zeroized when freeing the cipher handle
Diffie-Hellman public and private key	Shared secret computation.	Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800- 90A DRBG.	Either in plaintext or Encrypted through key wrapping using FC_WrapKey.	Key lengths 2048 bits and 15360 bits (or more)	RAM	Automatically zeroized when freeing the cipher handle
EC Diffie-Hellman public and private key	Shared secret computation.	Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800- 90A DRBG.	Either in plaintext or Encrypted through key wrapping using FC_WrapKey.	Curves P-256, P-384, P-521	RAM	Automatically zeroized when freeing the cipher handle
TLS pre-master secret	Establishment of encrypted session as input to the derivation of the master secret.	Generated during the shared secret computation when using Diffie-Hellman or EC Diffie-Hellman key exchange. Generated by TLS client as output from DRBG when using RSA key exchange.	Entry: if received by module as TLS server, wrapped with server's public RSA key; otherwise no entry. Output: if generated by module as TLS client, wrapped with server's public RSA key; otherwise, no output.	Length defined by ciphersuite (application)	RAM	Automatically zeroized when freeing the cipher handle
TLS Master secret	Establishment of encrypted session.	Derived from pre-master secret.	N/A	384 bits	RAM	Automatically zeroized when freeing the cipher handle

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Name	Use	Generation/ Establishment	Entry/ Output	Type	Storage	Zeroization
Entropy input string	Entropy input strings used to construct the seed for the DRBG.	Obtained from NDRNG.	N/A	384 bits	RAM	Automatically zeroized when freeing the DRBG handle
DRBG Internal state (V, C)	Used internally by DRBG. Used to generate random bits.	During DRBG initialization.	N/A	Internal state values	RAM	Automatically zeroized when freeing the DRBG handle
User Password	User authentication	Supplied by the calling application.	Entry: received from the calling application through API parameters. No output.	See Section 4.2.1	RAM or key data base in salted form	Automatically zeroized when the module is reinitialized or overwritten when the user changes its password
AES derived key	Encryption, decryption.	Generated internally by the module (from the [SP800-135] TLS KDF).	No entry. Output either in plaintext or Encrypted through key wrapping using FC_WrapKey.	AES key with modes and lengths in Table 7	RAM	Automatically zeroized when freeing the cipher handle
Triple-DES derived key	Encryption, decryption	Generated internally by the module (from the [SP800-135] TLS KDF).	No entry. Output either in plaintext or Encrypted through key wrapping using FC_WrapKey.	Triple-DES key with modes and lengths in Table 7	RAM	Automatically zeroized when freeing the cipher handle
HMAC derived key	MAC generation and verification	Generated internally by the module (from the [SP800-135] TLS KDF).	No entry. Output either in plaintext or Encrypted through key wrapping using FC_WrapKey.	HMAC key with lengths in Table 7	RAM	Automatically zeroized when freeing the cipher handle
TLS KDF internal state	Values of the TLS KDF	SP800-135 TLS KDF	N/A	Internal state values	RAM	Automatically zeroized when

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Name	Use	Generation/ Establishment	Entry/ Output	Type	Storage	Zeroization
	internal state					freeing the KDF handle

7.1 Random Number Generation

The module provides a DRBG compliant with [SP800-90A] for the creation of key components of asymmetric keys, and random number generation. The DRBG implements a HASH_DRBG mechanism with SHA-256 without prediction resistance.

The DRBG is initialized during module initialization and seeded from the NDRNG from /dev/urandom. The NDRNG is provided by the operational environment (i.e., Linux RNG), which is within the module’s physical boundary but outside of the module’s logical boundary. The NDRNG provides at least 256 bits of entropy to the DRBG.

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

The module performs the DRBG health testing as specified in Section 11.3 of NIST SP800-90A. The underlying operating system performs the continuous test on the NDRNG.

7.2 Key Generation

For generating RSA, DSA, ECDSA, Diffie-Hellman and EC Diffie-Hellman keys, the module implements asymmetric key generation services compliant with [FIPS186-4] and using a DRBG compliant with [SP800-90A]. The random value used in asymmetric key generation is obtained from the DRBG. In accordance with FIPS 140-2 IG D.12, the cryptographic module performs Cryptographic Key Generation (CKG) for asymmetric keys as per SP800-133 (vendor affirmed).

Symmetric keys are derived from the shared secret established by Diffie-Hellman and EC Diffie-Hellman in a manner that is compliant to NIST SP 800-135 for TLS KDF. The module also generates symmetric key through the FC_GenerateKey() function using the random numbers from the SP 800-90A DRBG.

The public and private key pairs used in the Diffie-Hellman and EC Diffie-Hellman shared secret computation schemes are generated internally by the module using the same DSA and ECDSA key generation compliant with [FIPS186-4] and with [SP800-56A].

7.3 Key Entry and Output

The module does not support manual key entry or intermediate key generation output. In addition, the module does not produce key output outside its physical boundary. The keys can be entered or output from the module in either plaintext form via API parameters or encrypted via key wrapping using FC_WrapKey. In both the cases the keys enter/output from the module to and from the calling application only.

7.4 Key/CSP Storage

Public and private keys are provided to the module by the calling process and according to the methods in Table 10. The module does not perform persistent storage of keys. When keys and CSPs are stored as plaintext in volatile memory (RAM), the protection of these keys and CSPs is provided by the operating system enforcement of separation of address space.

The private key database (provided with the files key3.db/key4.db) mentioned in Table 10 is within the module's physical boundary but outside its logical boundary.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

7.5 Key/CSP Zeroization

The application using the module is responsible for calling the appropriate destruction functions from the API to zeroize keys and CSPs. The destruction functions then overwrite the memory occupied by keys with zeros and deallocates the memory with the proper method call.

A plaintext secret or private key is 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_CloseSession or FC_CloseAllSessions call.

7.6 Key Establishment

The module provides Diffie-Hellman and EC Diffie-Hellman shared secret computation. The module also provides AES key wrapping per [SP800-38F] and RSA key wrapping (encapsulation) using public key encryption and private key decryption primitives as allowed by [FIPS140-2_IG] D.9. The shared secret computation and key wrapping schemes may be used by an application implementing the TLS protocol as a use case.

The module provides one approved key transport method according to IG D.9. The method comprises using the approved key wrapping technique, AES-KW.

Table 7 and Table 8 specify the key sizes allowed in the FIPS mode of operation. According to “Table 2: Comparable strengths” in [SP800-57 the key sizes of key wrapping, transport, and shared secret computation (using the respective symmetric algorithm, RSA, Diffie-Hellman and EC Diffie-Hellman) provide the following security strengths:

- RSA key wrapping provides between 112 and 256 bits of encryption strength.
- Diffie-Hellman shared secret computation provides between 112 and 256 bits of encryption strength.
- EC Diffie-Hellman shared secret computation provides between 128 and 256 bits of encryption strength.
- AES-KW key establishment methodology provides between 128 and 256 bits of encryption strength.

7.7 Handling of Keys and CSPs between Modes of Operation

As observed in Section 2.6, the module does not share CSPs between the FIPS-approved mode of operation and the non-FIPS mode of operation.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

8 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The test platforms listed in Table 3 have been tested and found to conform to the EMI/EMC requirements specified by 47 Code of Federal Regulations, FCC PART 15, Subpart B, Unintentional Radiators, Digital Devices, Class A (i.e., Business use). These devices are designed to provide reasonable protection against harmful interference when the devices are operated in a commercial environment.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

9 Self-Tests

9.1 Power-on Self-Tests

The module performs power-up or power-on self-tests (POSTs) automatically during loading of the module by making use of default entry point (DEP). These POSTs ensure that the module is not corrupted and that the cryptographic algorithms work as expected. No operator intervention is necessary to run the POSTs. The self-tests cover different implementations depending on the availability of those implementations in the operational environment (e.g., AES-NI).

While the module is executing the POSTs, services are not available, and input and output are inhibited. The module is not available for use until successful completion of the POSTs.

The integrity of the module binary is verified using a DSA signature with 2048 bits and SHA-256 . The signature value is computed at build time and stored in the .chk files. The value is recalculated at runtime and compared against the stored value in the file. If the comparison succeeds, then the remaining POSTs (consisting of the algorithm-specific Known Answer Tests) are performed.

On successful completion of the all the power-on tests, the module becomes operational and crypto services are then available. If any of the power-up self-tests fail, the module enters the Error state. In the Error state, all output is inhibited and no cryptographic operation is allowed. 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 the Error state.

Table 11 details the self-tests that are performed on the FIPS-approved cryptographic algorithms supported in the FIPS-approved mode of operation, using the Known-Answer Tests (KATs).

Table 11: Self-tests.

Algorithm	Test
AES	• KAT AES-CBC with 128-bit key, encryption • KAT AES-ECB with 128-bit key, decryption • KAT AES-CBC with 192-bit key, encryption • KAT AES-ECB with 192-bit key, decryption • KAT AES-CBC with 256-bit key, encryption • KAT AES-ECB with 256-bit key, decryption
Triple-DES	• KAT Triple-DES (CBC) with 192-bit key, encryption • KAT Triple-DES (CBC) with 192-bit key, decryption • KAT Triple-DES (ECB) with 192-bit key, encryption • KAT Triple-DES (ECB) with 192-bit key, decryption
DSA	• KAT DSA with 2048-bit key and SHA-256
RSA	• KAT RSA encryption and decryption with 2048-bit key • KAT RSA signature generation and verification with 2048-bit key and SHA-256, SHA-384, and SHA-512
ECDSA	• KAT ECDSA with P-256 and SHA-256
DRBG	• KAT HASH_DRBG using AES-256 without PR
HMAC	• KAT HMAC-SHA-1 • KAT HMAC-SHA-224

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Algorithm	Test
	• KAT HMAC-SHA-256 • KAT HMAC-SHA-384 • KAT HMAC-SHA-512
SHS	• KAT SHA-1 • KAT SHA-224 • KAT SHA-256 • KAT SHA-384 • KAT SHA-512
Module Integrity	• DSA signature with 2048 bits and SHA-256

9.2 Conditional Self-Tests

Conditional tests are performed during operational state of the module when the respective crypto functions are used. If any of the conditional tests fails, module transitions to 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 the Error state.

Table 12 lists the conditional self-tests performed by the functions.

Table 12: Conditional self-tests.

Algorithm	Test
DSA Key generation	PCT, signature generation and verification
ECDSA Key generation	PCT, signature generation and verification
RSA Key generation	PCT, signature generation and verification, and for encryption and decryption

The module performs the DRBG health testing as specified in Section 11.3 of [SP800-90A]. The CRNGT on the [SP800-90A] DRBG is not required per IG 9.8 [FIPS140-2_IG].

9.3 On-Demand self-tests

The module provides the Self-Test service to perform self-tests on demand. On demand self-tests can be invoked by powering-off and reloading the module. This service performs the same cryptographic algorithm tests executed during power-on. During the execution of the on-demand self-tests, cryptographic services are not available and no data output or input is possible.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

10 Guidance

This section provides guidance for the Crypto Officer and the User to maintain proper use of the module per FIPS 140-2 requirements.

10.1 Debug and Trace

As stated in Section 6.2, in operational mode, the ptrace(2) system call, the gdb(1) debugger and strace(1) shall not be used. In addition, other tracing mechanisms offered by the Linux environment, such as ftrace or systemtap, shall not be used.

10.2 Crypto-Officer Guidance

The binaries of the module are delivered via Red Hat Package Manager (RPM) packages. The Crypto Officer shall follow this Security Policy to configure the operational environment and install the module to be operated as FIPS 140-2 validated module. The version of the RPM packages containing the FIPS validated module are listed in Section 2.3.

To configure the operating environment to support FIPS perform the following steps:

1. Install the dracut-fips package:

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

2. Recreate the INITRAMFS image:

   ```
   # dracut -f
   ```

After regenerating the initramfs, the Crypto Officer must append the following string to the kernel command line by changing the setting in the boot loader:

```
fips=1
```

If /boot or /boot/efi reside on a separate partition, the kernel parameter boot=<partition of /boot or /boot/efi> must be supplied. The partition can be identified with the following command, respectively:

```
"df /boot"
```

or

```
"df /boot/efi"
```

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"
```

When supporting other formats such as boot=UUID/LABEL, please refer to the FIPS section of the 'dracut.cmdline' man page.

Reboot to apply above settings.

After performing the above configuration, the Crypto Officer should proceed to module installation. The RPM package of the module can be installed using standard tools recommended for the

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

installation of packages on an Amazon Linux 2 system (e.g., yum, RPM). The integrity of the RPM is automatically verified during the installation of the module and the Crypto Officer shall not install the RPM file if the yum server indicates an integrity error.

In addition, to support the module, the NSPR library must be installed that is offered by the underlying operating system.

10.2.1 Access to Audit Data

The module may use the Unix syslog function and the audit mechanism provided 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 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/log directory. The exact location of the system log is specified in the /etc/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.

10.3 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 variable 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 function 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 using the function pointer C_Logout, 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. And the Crypto-Officer role can be implicitly assumed by performing the Crypto-Officer services as listed in Section 4.3.2.

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 not be used by more than one process concurrently. “key4.db” format is based on SQL Database 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 to be stored in plaintext. The interface code of the module that accesses data stored in the database is considered part of the cryptographic boundary.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

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 (wrapped) form with FC_WrapKey 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.

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.

10.3.1 TLS Protocol

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.

10.3.2 Triple-DES Data Encryption

Data encryption using the same three-key Triple-DES key shall not exceed 2^{16} Triple-DES (64-bit) blocks, in accordance to [SP800-67] and IG A.13 in [FIPS140-2-IG].

10.3.3 Key Usage and Management

In general, a single key shall be used for only one purpose (e.g., encryption, integrity, authentication, key wrapping, random bit generation, or digital signatures) and be disjoint between the modes of operations of the module. Thus, if the module is switched between its FIPS mode and non-FIPS mode or vice versa, the following procedures shall be observed:

- The DRBG engine shall be reseeded.
- CSPs and keys shall not be shared between security functions of the two different modes.

The DRBG shall not be used for key generation for non-approved services in the non-FIPS mode.

10.4 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, or by power-cycling (power-off, power-on) the module.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

11 Mitigation of Other Attacks

The module is designed to mitigate the attacks as described next.

11.1.1 Timing Attacks on RSA

The mitigation mechanism for mitigation of this attack is RSA blinding.

Timing attack on RSA was first demonstrated by [Kocher, 1996], who contributed the mitigation code to our module. Most recently [Boneh; Brumley, 2019] showed that RSA blinding is an effective defense against timing attacks on RSA.

11.1.2 Cache-Timing Attacks on RSA and DSA

These cache-timing attacks target the modular exponentiation operation used in RSA an DSA algorithms.

The mitigation mechanism to mitigate this attack is the “cache invariant modular exponentiation”. This is a variant of a modular exponentiation implementation that [Percival, 2019] 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 cache line sizes are unknown.

11.1.3 Arithmetic Errors in RSA Signatures

This attack is based on the fact that arithmetic errors in RSA signatures might leak the private key. The mitigation technique for this attack is “Double-Checking RSA Signatures”.

[Ferguson; Schneier, 2003] recommend that every RSA signature generation should verify the signature just generated.

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

12 Acronyms, Terms and Abbreviations

Term	Definition
AES	Advanced Encryption Standard
AESNI	Advanced Encryption Standard New Instructions
CAVP	Cryptographic Algorithm Validation Program
CMVP	Cryptographic Module Validation Program
CSE	Communications Security Establishment
CSP	Critical Security Parameter
DANE	DNS-based Authentication of Named Entities
DH	Diffie-Hellman
DHE	Diffie-Hellman Ephemeral
DRBG	Deterministic Random Bit Generator
DTLS	Datagram Transport Layer Security
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EDC	Error Detection Code
GCM	Galois-Counter Mode
HMAC	(Keyed) Hash Message Authentication Code
IKE	Internet Key Exchange
KAT	Known Answer Test
KDF	Key Derivation Function
NDRNG	Non-Deterministic Random Number generator
NIST	National Institute of Standards and Technology
PAA	Processor Algorithm Acceleration
PKCS	Public Key Cryptography Standard
POST	Power On Self-Test
PR	Prediction Resistance
PSS	Probabilistic Signature Scheme
PUB	Publication
SHA	Secure Hash Algorithm
TLS	Transport Layer Security

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

13 References

Boneh; Brumley	**Remote Timing Attacks are Practical** Accessed 2019-Jun-13 https://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf
Ferguson; Schneier	**Practical Cryptography.** Wiley Publishing, Inc. 2003
FIPS140-2	**FIPS PUB 140-2 - Security Requirements for Cryptographic Modules** May 2001 http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
FIPS140-2_IG	**Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program** May 7, 2019 https://csrc.nist.gov/CSRC/media/Projects/cryptographic-module-validation-program/documents/fips140-2/FIPS1402IG.pdf
FIPS180-4	**Secure Hash Standard (SHS)** March 2012 http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
FIPS186-4	**Digital Signature Standard (DSS** July 2013 http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS197	**Advanced Encryption Standard** November 2001 http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
FIPS198-1	**The Keyed Hash Message Authentication Code (HMAC)** July 2008 http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
Kocher	**Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other Systems.** CRYPTO '96, Lecture Notes in Computer Science, v. 1109, pp. 104-113. Springer-Verlag 1996 http://www.cryptography.com/timingattack/
Percival	**Cache Missing for Fun and Profit** Accessed 2019-Jun-13 http://www.daemonology.net/papers/htt.pdf
PKCS#1	**Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.2** November 2016 https://tools.ietf.org/rfc/rfc8017.txt

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

PKCS#11 **PKCS#11 v2.40: Cryptographic Token Interface Standard**
April 2015
https://www.oasis-open.org/standards#pkcs11-base-v2.40

RFC3711 **The Secure Real-time Transport Protocol (SRTP)**
March 2004
https://tools.ietf.org/html/rfc3711

RFC4347 **Datagram Transport Layer Security**
April 2006
https://tools.ietf.org/html/rfc4347

RFC4357 **Additional Cryptographic Algorithms for Use with GOST 28147-89, GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms**
January 2006
https://tools.ietf.org/html/rfc4357

RFC5246 **The Transport Layer Security (TLS) Protocol Version 1.2**
August 2008
https://tools.ietf.org/html/rfc5246

RFC5288 **AES Galois Counter Mode (GCM) Cipher Suites for TLS**
August 2008
https://tools.ietf.org/html/rfc5288

RFC5764 **Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)**
May 2010
https://tools.ietf.org/html/rfc5764

SP800-131A **NIST Special Publication 800-131A Revision 2- Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths**
March 2019
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf

SP800-135 **NIST Special Publication 800-135 Revision 1 - Recommendation for Existing Application-Specific Key Derivation Functions**
December 2011
http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf

SP800-38A **NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques**
December 2001
http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

SP800-52 **NIST Special Publication 800-52 Revision 1 - Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations**
April 2014
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

SP800-56A	**NIST Special Publication 800-56A - Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)** March, 2007 http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf
SP800-67	**NIST Special Publication 800-67 Revision 2 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher** November 2017 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf
SP800-90A	**NIST Special Publication 800-90A - Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** June 2015 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

© 2020 Amazon Web Services, Inc.; atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.