Red Hat Enterprise Linux 9 OpenSSL FIPS Provider

version 3.0.1-3f45e68ee408cd9c

FIPS 140-3 Non-Proprietary Security Policy

document version 1.2

Last update: 2024-06-14

Prepared by:

atsec information security corporation

4516 Seton Center Parkway, Suite 250

Austin, TX 78759

www.atsec.com

Table of Contents

1 General

1.1 Overview

This document is the non-proprietary FIPS 140-3 Security Policy for version 3.0.1-3f45e68ee408cd9c of the Red Hat Enterprise Linux 9 OpenSSL FIPS Provider. It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS PUB 140-3 (Federal Information Processing Standards Publication 140-3) for an overall Security Level 1 module.

This Non-Proprietary Security Policy may be reproduced and distributed, but only whole and intact and including this notice. Other documentation is proprietary to their authors.

1.2 How this Security Policy was prepared

The vendor has provided the non-proprietary Security Policy of the cryptographic module, which was further consolidated into this document by atsec information security together with other vendor-supplied documentation. In preparing the Security Policy document, the laboratory formatted the vendor-supplied documentation for consolidation without altering the technical statements therein contained. The further refining of the Security Policy document was conducted iteratively throughout the conformance testing, wherein the Security Policy was submitted to the vendor, who would then edit, modify, and add technical contents. The vendor would also supply additional documentation, which the laboratory formatted into the existing Security Policy, and resubmitted to the vendor for their final editing.

1.3 Security levels

Table 1 describes the individual security areas of FIPS 140-3, as well as the security levels of those individual areas.

ISO/IEC 24759 Section 6. [Number Below]	FIPS 140-3 Section Title	Security Level
1	General	1
2	Cryptographic Module Specification	1
3	Cryptographic Module Interfaces	1
4	Roles, Services, and Authentication	1
5	Software/Firmware Security	1
6	Operational Environment	1
7	Physical Security	Not Applicable
8	Non-invasive Security	Not Applicable
9	Sensitive Security Parameter Management	1
10	Self-tests	1

11	Life-cycle Assurance	1
12	Mitigation of Other Attacks	1

Table 1 - Security Levels

2 Cryptographic module specification

2.1 Description

The Red Hat Enterprise Linux 9 OpenSSL FIPS Provider (hereafter referred to as "the module") is defined as a software module in a multi-chip standalone embodiment. It provides a C language application program interface (API) for use by other applications that require cryptographic functionality. The module consists of one software component, the "FIPS provider", which implements the FIPS requirements and the cryptographic functionality provided to the operator.

2.2 Operational environments

The module has been tested on the following platforms with the corresponding module variants and configuration options with and without PAA:

#	Operating System	Hardware Platform	Processor	PAA/ Acceleration
1	Red Hat Enterprise Linux 9	Dell PowerEdge R440	Intel(R) Xeon(R) Silver 4216	AES-NI, SHA extensions
2	Red Hat Enterprise Linux 9	IBM z16 3931-A01	IBM z16	CPACF
3	Red Hat Enterprise Linux 9	IBM 9080 HEX	IBM POWER10	ISA

Table 2 - Tested Operational Environments

In addition to the configurations tested by the atsec CST laboratory, the vendor affirms testing was performed on the following platforms for the module.

#	Operating System	Hardware Platform
1	Red Hat Enterprise Linux 9	Intel(R) Xeon(R) E5

Table 3 - Vendor Affirmed Operational Environments

Note: the CMVP makes no statement as to the correct operation of the module or the security strengths of the generated SSPs when so ported if the specific operational environment is not listed on the validation certificate.

2.3 Approved algorithms

Table 4 lists all approved cryptographic algorithms of the module, including specific key lengths employed for approved services (Table 9), and implemented modes or methods of operation of the algorithms.

The module supports RSA modulus sizes which are not tested by CAVP in compliance with FIPS 140-3 IG C.F.

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A3544 A3545 A3546 A3547 A4022 A4459	SHA [FIPS 180-4]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256	N/A	Message digest
A3534 A4024	SHA-3 [FIPS 202]	SHA3-224, SHA3-256, SHA3-384, SHA3-512	N/A	Message digest
	SHA-3 [FIPS 202]	SHAKE128, SHAKE256	N/A	XOF
A3527 A3528 A3529 A4018 A4023 A4460 A4461 A4465	AES [FIPS 197, SP 800-38A, SP 800-38A Addendum, SP 800-38C, SP 800-38F]	ECB, CBC, CBC-CTS-CS1, CBC-CTS-CS2, CBC-CTS-CS3, CFB1, CFB8, CFB128, CTR, OFB, CCM KW, KWP (KTS)	128, 192, 256 bits with 128, 192, 256 bits of security strength	Encryption Decryption
A3535 A3536 A3537 A3538 A3539 A3540 A3541 A3542 A3543 A4019 A4020 A4021 A4458 A4462	AES [FIPS 197, SP 800-38D]	GCM (internal IV)	128, 192, 256 bits with 128, 192, 256 bits of security strength	Encryption
	AES [FIPS 197, SP 800-38D]	GCM (external IV)	128, 192, 256 bits with 128, 192, 256 bits of security strength	Decryption
A3527 A3528 A3529 A4018 A4461 A4465	AES [FIPS 197, SP 800-38E]	XTS	128, 256 bits with 128, 256 bits of security strength	Encryption Decryption
	AES [FIPS 197, SP 800-38B]	CMAC	128, 192, 256 bits with 128, 192, 256 bits of security strength	Message authentication

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A3535 A3536 A3537 A3538 A3539 A3540 A3541 A3542 A3543 A4019 A4020 A4021 A4458 A4462	AES [FIPS 197, SP 800-38D]	GMAC	128, 192, 256 bits with 128, 192, 256 bits of security strength	Message authentication
A3544 A3545 A3546 A3547 A4022 A4459	HMAC [FIPS 198-1]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256	112-524288 bits with 112-256 bits of security strength	Message authentication
A3534 A4024		SHA3-224, SHA3-256, SHA3-384, SHA3-512		
A3553	KBKDF [SP 800-108r1]	Counter and feedback mode, using CMAC and HMAC SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	112-4096 bits with 112-256 bits of security strength	KBKDF Key derivation
A3525	KDA OneStep[1] [SP 800-56Cr2]	(HMAC) SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	224-8192 bits with 112-256 bits of security strength	KDA OneStep Key derivation
A3526	HKDF [SP 800-56Cr2]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	224-8192 bits with 112-256 bits of security strength	HKDF Key derivation
A3534 A3544 A3545 A3546 A3547 A4022	ANS X9.42 KDF [SP 800-135r1] CVL	AES KW with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	224-8192 bits with 112-256 bits of security strength	ANS X9.42 KDF Key derivation

[1] This algorithm is referred to as “Single Step KDF” or “SSKDF” by OpenSSL.

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A4024 A4459	ANS X9.63 KDF [SP 800-135r1] CVL	SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	224-8192 bits with 112-256 bits of security strength	ANS X9.63 KDF Key derivation
A3530 A3531 A3532 A3533 A4023 A4460	SSH KDF [SP 800-135r1] CVL	AES-128, AES-192, AES-256 with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	224-8192 bits with 112-256 bits of security strength	SSH KDF Key derivation
A3544 A3545 A3546 A3547 A4022 A4459	TLS 1.2 KDF [SP 800-135r1] CVL	SHA-256, SHA-384, SHA-512	224-8192 bits with 112-256 bits of security strength	TLS 1.2 KDF Key derivation
A3526	TLS 1.3 KDF [RFC 8446] CVL	SHA-256, SHA-384	224-8192 bits with 112-256 bits of security strength	TLS 1.3 KDF Key derivation
A3534 A3544 A3545 A3546 A3547 A4022 A4024 A4459	PBKDF2 [SP 800-132]	Option 1a with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	8-128 characters with password strength between 10^8 and 10^{128}	Password-based key derivation
A3570	CTR_DRBG [SP 800-90Ar1]	AES-128, AES-192, AES-256, with/without derivation function, with/without prediction resistance	256, 320, 384 bits with 128, 192, 256 bits of security strength	Random number generation
A3570	Hash_DRBG [SP 800-90Ar1]	SHA-1, SHA-256, SHA-512 with/without prediction resistance	880, 1776 bits with 128, 256 bits of security strength	Random number generation
A3570	HMAC_DRBG [SP 800-90Ar1]	SHA-1, SHA-256, SHA-512 with/without prediction resistance	320, 512, 1024 bits with 128, 256 bits of security strength	Random number generation
A3554	KAS-FFC-SSC [SP 800-56Ar3]	dhEphem (initiator/responder)	MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192, ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192 with 112-200 bits of security strength	Shared secret computation

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A3544 A3545 A3546 A3547 A4022 A4459	KAS-ECC-SSC [SP 800-56Ar3]	Ephemeral Unified Model (initiator/responder)	P-224, P-256, P-384, P-521 with 112, 128, 192, 256 bits of security strength	Shared secret computation
	RSA [FIPS 186-4]	PKCS#1 v1.5 and PSS with SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	2048-16384 bits with 112-256 bits of security strength	Signature generation
	RSA [FIPS 186-4]		NIST SP 800-131Ar2 Legacy use: 1024-2047 bits with 80-111 bits of security strength NIST SP 800-131Ar2 Acceptable: 2048-16384 bits with 112-256 bits of security strength	Signature verification
A3534 A3544 A3545 A3546 A3547 A4022 A4024 A4459	ECDSA [FIPS 186-4]	SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	P-224, P-256, P-384, P-521 with 112, 128, 192, 256 bits of security strength	Signature generation
	ECDSA [FIPS 186-4]			Signature verification
A3554	Safe primes [SP 800-56Ar3]	SP 800-56Ar3 Section 5.6.1.1.4 Testing Candidates	MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192, ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192 with 112-200 bits of security strength	Key pair generation
A3554	Safe primes [SP 800-56Ar3]	SP 800-56Ar3 Sections 5.6.2.1.2 and 5.6.2.1.4		Key pair verification
A3544 A3545 A3546 A3547 A4022 A4459	RSA [FIPS 186-4]	FIPS 186-4 Appendix B.3.6 Probable Primes with Conditions Based on Auxiliary Probable Primes	2048-15360 bits with 112-256 bits of security strength	Key pair generation
	ECDSA [FIPS 186-4]	FIPS 186-4 Appendix B.4.2 Testing Candidates	P-224, P-256, P-384, P-521 with 112, 128, 192, 256 bits of security strength	Key pair generation
	ECDSA [FIPS 186-4]	N/A		Key pair verification
Vendor affirmed	CKG [SP 800-133r2 Section 4]	Safe primes	MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192, ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192 with 112-200 bits of security strength	Key pair generation
		RSA	2048-16384 bits with 112-256 bits of security strength	

CAVP Cert	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
		ECDSA	P-224, P-256, P-384, P-521 with 112, 128, 192, 256 bits of security strength	
Vendor affirmed	RSA [FIPS 186-4] SHA-3 [FIPS 202] [FIPS 140-3 IG C.C]	RSA PKCS#1 v1.5 and PSS with SHA3-224, SHA3-256, SHA3-384, SHA3-512	2048-16384 bits with 112-256 bits of security strength	Signature generation
			NIST SP 800-131Ar2 Legacy use: 1024-2047 bits with 80-111 bits of security strength NIST SP 800-131Ar2 Acceptable: 2048-16384 bits with 112-256 bits of security strength	Signature verification

Table 4 - Approved Algorithms

2.4 Non-approved algorithms

The module does not offer any non-approved cryptographic algorithms that are allowed in approved services (with or without security claimed).

Table 5 lists all non-approved cryptographic algorithms of the module employed by the non-approved services in Table 10.

Algorithm / Functions	Use / Function
AES GCM (external IV)	Encryption
HMAC (< 112-bit keys)	Message authentication
KBKDF, KDA OneStep, HKDF, ANS X9.42 KDF, ANS X9.63 KDF (< 112-bit keys)	KBKDF Key derivation KDA OneStep Key derivation HKDF Key derivation ANS X9.42 KDF Key derivation ANS X9.63 KDF Key derivation
KDA OneStep (SHAKE128, SHAKE256)	KDA OneStep Key derivation
ANS X9.42 KDF (SHAKE128, SHAKE256)	ANS X9.42 KDF Key derivation
ANS X9.63 KDF (SHA-1, SHAKE128, SHAKE256)	ANS X9.63 KDF Key derivation
SSH KDF (SHA-512/224, SHA-512/256, SHA-3, SHAKE128, SHAKE256)	SSH KDF Key derivation
TLS 1.2 KDF (SHA-1, SHA-224, SHA-512/224, SHA-512/256, SHA-3)	TLS 1.2 KDF Key derivation
TLS 1.3 KDF (SHA-1, SHA-224, SHA-512, SHA-512/224, SHA-512/256, SHA-3)	TLS 1.3 KDF Key derivation

PBKDF2 (short password; short salt; insufficient iterations; < 112-bit keys)	Password-based key derivation
KAS1, KAS2	Shared secret computation
RSA and ECDSA (pre-hashed message)	Signature generation Signature verification
RSA-PSS (invalid salt length)	
RSA-OAEP	Asymmetric encryption Asymmetric decryption

Table 5 - Non-Approved Algorithms Not Allowed in the Approved Mode of Operation

2.5 Module design and components

Figure 1 shows a block diagram that represents the design of the module when the module is operational and providing services to other user space applications. In this diagram, the physical perimeter of the operational environment (a general-purpose computer on which the module is installed) is indicated by a purple dashed line. The cryptographic boundary is represented by the component painted in orange block, which consists only of the shared library implementing the FIPS provider (fips.so).

Green lines indicate the flow of data between the cryptographic module and its operator application, through the logical interfaces defined in Section 3.

Components in white are only included in the diagram for informational purposes. They are not included in the cryptographic boundary (and therefore not part of the module's validation). For example, the kernel is responsible for managing system calls issued by the module itself, as well as other applications using the module for cryptographic services.

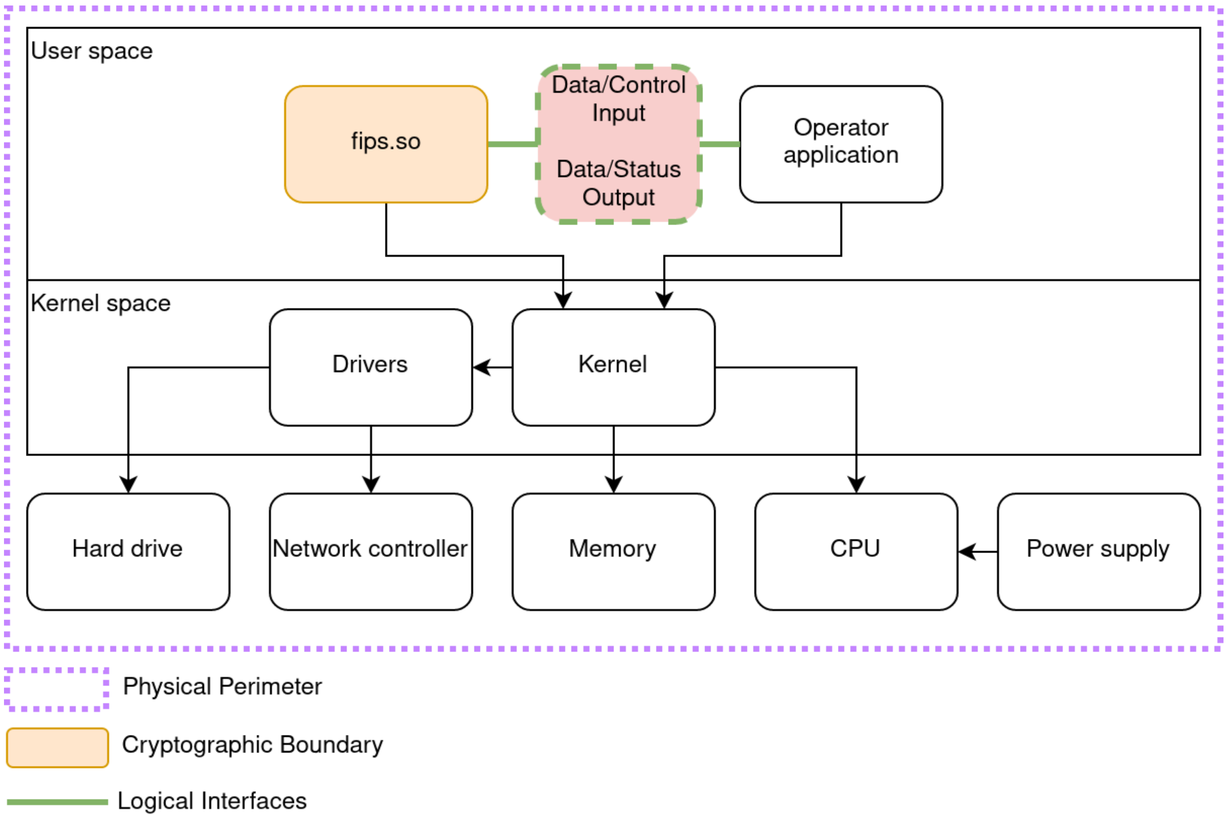

Figure 1 – Software Block Diagram

2.6 Rules of operation

Upon initialization, the module immediately performs all cryptographic algorithm self-tests (CASTs) as specified in Table 13. When all those self-tests pass successfully, the module automatically performs the pre-operational integrity test using the integrity value embedded in the fips.so file. Only if this integrity test also passed successfully, the module transitions to the operational state. No operator intervention is required to reach this point. The module operates in the approved mode of operation by default and can only transition into the non-approved mode by calling one of the non-approved services listed in Table 10 of the Security Policy.

In the operational state, the module accepts service requests from calling applications through its logical interfaces. At any point in the operational state, a calling application can end its process, thus causing the module to end its operation.

The module supports two modes of operation:

- The approved mode of operation, in which the approved or vendor affirmed services are available as specified in Table 9.
- The non-approved mode of operation, in which the non-approved services are available as specified in Table 10.

3 Cryptographic module interfaces

The logical interfaces are the APIs through which the applications request services. These logical interfaces are logically separated from each other by the API design. Table 6 summarizes the logical interfaces:

Physical Port	Logical Interface	Data that passes over port / interface
As a software-only module, the module does not have physical ports. Physical Ports are interpreted to be the physical ports of the hardware platform on which it runs.	Data Input	API input parameters
	Data Output	API output parameters
	Control Input	API function calls
	Status Output	API return codes, error queue

Table 6 - Ports and Interfaces

The module does not implement a control output interface.

4 Roles, services, and authentication

4.1 Roles

The module supports the Crypto Officer role only. This sole role is implicitly and always assumed by the operator of the module. No support is provided for multiple concurrent operators or a maintenance role.

Table 7 lists the roles supported by the module with corresponding services with input and output parameters.

Role	Service	Input	Output
Crypto Officer	Message digest	Message	Digest value
	XOF	Message, output length	Digest value
	Encryption	Plaintext, AES key	Ciphertext
	Decryption	Ciphertext, AES key	Plaintext
	Message authentication	Message, AES key or HMAC key	MAC tag
	KBKDF Key derivation	Key-derivation key	KBKDF Derived key
	KDA OneStep Key derivation	Shared secret	KDA OneStep Derived key
	HKDF Key derivation	Shared secret	HKDF Derived key
	ANS X9.42 KDF Key derivation	Shared secret	ANS X9.42 KDF Derived key
	ANS X9.63 KDF Key derivation	Shared secret	ANS X9.63 KDF Derived key
	SSH KDF Key derivation	Shared secret	SSH KDF Derived key
	TLS 1.2 KDF Key derivation	Shared secret	TLS 1.2 KDF Derived key
	TLS 1.3 KDF Key derivation	Shared secret	TLS 1.3 KDF Derived key
	Password-based key derivation	Password, salt, iteration count	PBKDF2 Derived key
	Random number generation	Output length	Random bytes
	Shared secret computation	Owner private key, peer public key	Shared secret
	Signature generation	Message, private key	Signature
	Signature verification	Message, public key, signature	Pass/fail
	Asymmetric encryption	Plaintext, public key	Ciphertext
	Asymmetric decryption	Ciphertext, private key	Plaintext
	Key pair generation	Key size	Key pair
	Key pair verification	Key pair	Pass/fail

	Show version	N/A	Name and version information
	Show status	N/A	Module status
	Self-test	N/A	Pass/fail results of self-tests
	Zeroization	Any SSP	N/A

Table 7 - Roles, Service Commands, Input and Output

4.2 Authentication

The module does not support authentication for roles.

4.3 Services

The module provides services to operators that assume the available role. All services are described in detail in the API documentation (manual pages). The next tables define the services that utilize approved and non-approved security functions in this module. For the respective tables, the convention below applies when specifying the access permissions (types) that the service has for each SSP.

- **Generate (G)**: The module generates or derives the SSP.
- **Read (R)**: The SSP is read from the module (e.g., the SSP is output).
- **Write (W)**: The SSP is updated, imported, or written to the module.
- **Execute (E)**: The module uses the SSP in performing a cryptographic operation.
- **Zeroize (Z)**: The module zeroizes the SSP.
- **N/A**: The module does not access any SSP or key during its operation.

To interact with the module, a calling application must use the EVP API layer provided by OpenSSL. This layer will delegate the request to the FIPS provider, which will in turn perform the requested service. Additionally, this EVP API layer can be used to retrieve the approved service indicator for the module. The `redhat_ossl_query_fipsindicator()` function indicates whether an EVP API function is approved. After a cryptographic service was performed by the module, the API context associated with this request can contain a parameter (listed in Table 8) which represents the approved service indicator.

Context	Service Indicator
EVP_CIPHER_CTX	OSSL_CIPHER_PARAM_REDHAT_FIPS_INDICATOR
EVP_MAC_CTX	OSSL_MAC_PARAM_REDHAT_FIPS_INDICATOR
EVP_KDF_CTX	OSSL_KDF_PARAM_REDHAT_FIPS_INDICATOR
EVP_PKEY_CTX	OSSL_SIGNATURE_PARAM_REDHAT_FIPS_INDICATOR
EVP_PKEY_CTX	OSSL_ASYM_CIPHER_PARAM_REDHAT_FIPS_INDICATOR

Table 8 - Service Indicator Parameters

The details to use these functions and parameters are described in the module’s manual pages.

Table 9 lists the approved services in this module, the algorithms involved, the Sensitive Security Parameters (SSPs) involved and how they are accessed, the roles that can request the service, and the respective service indicator. In this table, CO specifies the Crypto Officer role.

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Message digest	Compute a message digest	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	N/A	CO	N/A	EVP_DigestFinal_ex returns 1
XOF	Compute the output of an XOF	SHAKE128, SHAKE256	N/A	CO	N/A	EVP_DigestFinalXOF returns 1
Encryption	Encrypt a plaintext	AES ECB, CBC, CBC-CTS-CS1, CBC-CTS-CS2, CBC-CTS-CS3, CFB1, CFB8, CFB128, CTR, OFB, CCM, KW, KWP, GCM, XTS	AES key	CO	W, E	AES GCM: EVP_CIPHER_REDHAT_FIPS_INDICATOR_APPROVED Others: EVP_EncryptFinal_ex returns 1
Decryption	Decrypt a ciphertext			CO	W, E	AES GCM: EVP_CIPHER_REDHAT_FIPS_INDICATOR_APPROVED Others: EVP_DecryptFinal_ex returns 1
Message authentication	Compute a MAC tag	AES CMAC AES GMAC HMAC SHA-1, HMAC SHA-224, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512, HMAC SHA-512/224, HMAC SHA-512/256, HMAC SHA3-224, HMAC SHA3-256, HMAC SHA3-384, HMAC SHA3-512	AES key HMAC key	CO	W, E	HMAC: OSSL_MAC_PARAM_REDHAT_FIPS_INDICATOR_APPROVED Others: EVP_MAC_final returns 1
KBKDF Key derivation	Derive a key from a key-derivation key	KBKDF	Key-derivation key	CO	W, E	EVP_KDF_REDHAT_FIPS_INDICATOR_APPROVED
			KBKDF Derived key		G, R	
KDA OneStep Key derivation	Derive a key from a shared secret	KDA OneStep	DH Shared secret		W, E	
			ECDH Shared secret			
			KDA OneStep Derived key		G, R	
HKDF Key derivation		HKDF	DH Shared secret		W, E	
			ECDH Shared secret			

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
			HKDF Derived key		G, R	
ANS X9.42 KDF Key derivation		ANS X9.42 KDF	DH Shared secret		W, E	
			ECDH Shared secret			
			ANS X9.42 KDF Derived key		G, R	
ANS X9.63 KDF Key derivation		ANS X9.63 KDF	DH Shared secret		W, E	
			ECDH Shared secret			
			ANS X9.63 KDF Derived key		G, R	
SSH KDF Key derivation		SSH KDF	DH Shared secret		W, E	
			ECDH Shared secret			
			SSH KDF Derived key		G, R	
TLS 1.2 KDF Key derivation		TLS 1.2 KDF	DH Shared secret		W, E	
			ECDH Shared secret			
			TLS 1.2 KDF Derived key		G, R	
TLS 1.3 KDF Key derivation		TLS 1.3 KDF	DH Shared secret		W, E	
			ECDH Shared secret			
			TLS 1.3 KDF Derived key		G, R	
Password-based key derivation	Derive a key from a password	PBKDF2	Password	CO	W, E	EVP_KDF_REDHAT_FIPS_INDICATOR_APPROVED
			PBKDF2 Derived key		G, R	
Random number generation	Generate random bytes	CTR_DRBG	Entropy input	CO	W, E	EVP_RAND_generate returns 1
			DRBG seed		E, G	
			Internal state (V, Key)		W, E, G	
		Hash_DRBG	Entropy input		W, E	
			DRBG seed		E, G	
			Internal state (V, C)		W, E, G	
		HMAC_DRBG	Entropy input		W, E	
			DRBG seed		E, G	

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
			Internal state (V, Key)		W, E, G	
Shared secret computation	Compute a shared secret	KAS-FFC-SSC	DH private key (owner), DH public key (peer)	CO	W, E	EVP_PKEY_derive returns 1
			DH Shared secret		G, R	
		KAS-ECC-SSC	EC private key (owner), EC public key (peer)		W, E	
			ECDH Shared secret		G, R	
Signature generation	Generate a signature	RSA signature generation/verification (PKCS#1 v1.5 and PSS) ECDSA signature generation/verification	RSA private key EC private key	CO	W, E	RSA: OSSL_RH_FIPSINDICATOR_APPROVED and EVP_SIGNATURE_REDHAT_FIPS_INDICATOR_APPROVED ECDSA: OSSL_RH_FIPSINDICATOR_APPROVED
Signature verification	Verify a signature		RSA public key EC public key	CO	W, E	
Key pair generation	Generate a key pair	CKG CTR_DRBG, Hash_DRBG, HMAC_DRBG Safe primes key pair generation RSA key pair generation ECDSA key pair generation	DH private key, DH public key RSA private key, RSA public key EC private key, EC public key Intermediate key generation value	CO	G, R	EVP_PKEY_generate returns 1
Key pair verification	Verify a key pair	Safe primes key pair verification ECDSA key pair verification	DH private key, DH public key EC private key, EC public key	CO	W, E	EVP_PKEY_public_check or EVP_PKEY_private_check or EVP_PKEY_check returns 1
Show version	Return the name and version information	N/A	N/A	CO	N/A	None
Show status	Return the module status	N/A	N/A	CO	N/A	None

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Self-test	Perform the CASTs and integrity test	SHA-1, SHA-224, SHA-256, SHA-512, SHA3-256 AES ECB, KW, GCM HMAC KBKDF, KDA OneStep, HKDF, ANS X9.42 KDF, ANS X9.63 KDF, SSH KDF, TLS 1.2 KDF, TLS 1.3 KDF PBKDF2 CTR_DRBG, Hash_DRBG, HMAC_DRBG KAS-FFC-SSC, KAS-ECC-SSC RSA (PKCS#1 v1.5) ECDSA See Table 13 for specifics	AES key HMAC key Key-derivation key Password DH private key, DH public key RSA private key, RSA public key EC private key, EC public key	CO	E	None
			DH Shared secret ECDH Shared secret KBKDF Derived key KDA OneStep Derived key HKDF Derived key ANS X9.42 KDF Derived key ANS X9.63 KDF Derived key SSH KDF Derived key TLS 1.2 KDF Derived key TLS 1.3 KDF Derived key PBKDF2 Derived key DRBG seed Internal state (V, Key) Internal state (V, C)		E, G	
Zeroization	Zeroize all SSPs	N/A	Any SSP	CO	Z	None

Table 9 - Approved Services

Table 10 lists the non-approved services in this module, the algorithms involved, the roles that can request the service, and the respective service indicator. In this table, CO specifies the Crypto Officer role.

Service	Description	Algorithms Accessed	Role
Encryption	Encrypt a plaintext	AES GCM (external IV)	CO

Service	Description	Algorithms Accessed	Role
Message authentication	Compute a MAC tag	HMAC (< 112-bit keys)	CO
KBKDF Key derivation	Derive a key from a key-derivation key	KBKDF (< 112-bit keys)	CO
KDA OneStep Key derivation	Derive a key from a shared secret	KDA OneStep (< 112-bit keys) KDA OneStep (SHAKE128, SHAKE256)	
HKDF Key derivation		HKDF (< 112-bit keys)	
ANS X9.42 KDF Key derivation		ANS X9.42 KDF (< 112-bit keys) ANS X9.42 KDF (SHAKE128, SHAKE256)	
ANS X9.63 KDF Key derivation		ANS X9.63 KDF (< 112-bit keys) ANS X9.63 KDF (SHA-1, SHAKE128, SHAKE256)	
SSH KDF Key derivation		SSH KDF (< 112-bit keys) SSH KDF (SHA-512/224, SHA-512/256, SHA-3, SHAKE128, SHAKE256)	
TLS 1.2 KDF Key derivation		TLS 1.2 KDF (< 112-bit keys) TLS 1.2 KDF (SHA-1, SHA-224, SHA-512/224, SHA-512/256, SHA-3)	
TLS 1.3 KDF Key derivation		TLS 1.3 KDF (< 112-bit keys) TLS 1.3 KDF (SHA-1, SHA-224, SHA-512, SHA-512/224, SHA-512/256, SHA-3)	
Password-based key derivation	Derive a key from a password	PBKDF2 (short password; short salt; insufficient iterations; < 112-bit keys)	CO
Shared secret computation	Compute a shared secret	KAS1, KAS2	CO
Signature generation	Generate a signature	RSA and ECDSA signature generation/verification (pre-hashed message)	CO
Signature verification	Verify a signature		CO
Asymmetric encryption	Encrypt a plaintext	RSA-OAEP encryption/decryption	CO
Asymmetric decryption	Decrypt a plaintext		CO

Table 10 - Non-Approved Services

5 Software/Firmware security

5.1 Integrity techniques

The integrity of the module is verified by comparing a HMAC SHA-256 value calculated at run time with the HMAC SHA-256 value embedded in the fips.so file that was computed at build time.

5.2 On-demand integrity test

Integrity tests are performed as part of the pre-operational self-tests, which are executed when the module is initialized. The integrity test may be invoked on-demand by unloading and subsequently re-initializing the module. This will perform (among others) the software integrity test.

6 Operational environment

6.1 Applicability

The module operates in a modifiable operational environment per FIPS 140-3 level 1 specification: the module executes on a general purpose operating system (Red Hat Enterprise Linux 9), which allows modification, loading, and execution of software that is not part of the validated module.

6.2 Tested operational environments

See Section 2.2.

The Red Hat Enterprise Linux operating system is used as the basis of other products which include but are not limited to:

- Red Hat Enterprise Linux CoreOS
- Red Hat Ansible Automation Platform
- Red Hat OpenStack Platform
- Red Hat OpenShift
- Red Hat Gluster Storage
- Red Hat Satellite

Compliance is maintained for these products whenever the binary is found unchanged.

6.3 Policy and requirements

The module shall be installed as stated in Section 11. If properly installed, the operating system provides process isolation and memory protection mechanisms that ensure appropriate separation for memory access among the processes on the system. Each process has control over its own data and uncontrolled access to the data of other processes is prevented.

There are no concurrent operators.

The module does not have the capability of loading software or firmware from an external source.

Instrumentation tools like the `ptrace` system call, gdb and `strace`, userspace live patching, as well as other tracing mechanisms offered by the Linux environment such as `ftrace` or `systemtap`, shall not be used in the operational environment. The use of any of these tools implies that the cryptographic module is running in a non-validated operational environment.

7 Physical security

The module is comprised of software only and therefore this section is not applicable.

8 Non-invasive security

This module does not implement any non-invasive security mechanism and therefore this section is not applicable.

9 Sensitive security parameters management

Table 10 summarizes the Sensitive Security Parameters (SSPs) that are used by the cryptographic services implemented in the module in the approved services (Table 9).

SSPs (including CSPs) are directly imported as input parameters and exported as output parameters from the module. Because these SSPs are only transiently used for a specific service, they are by definition exclusive between approved and non-approved services.

Key / SSP Name / Type	Strength	Security Function and Cert. Number	Generation	Import / Export	Establishment	Storage	Zeroization	Use and related keys
AES key (CSP)	AES-XTS: 128, 256 bits Rest of modes: 128, 192, 256 bits	AES AES CMAC AES GMAC A3527, A3528, A3529, A3535, A3536, A3537, A3538, A3539, A3540, A3541, A3542, A3543, A4018, A4019, A4020, A4021, A4023, A4458, A4460, A4461, A4462, A4465	N/A	MD/EE Import: API input parameters From: Operator calling application (TOEPP) To: Cryptographic module Export: None	N/A	RAM	EVP_CIPHER_CTX_free EVP_MAC_CTX_free	Use: Encryption Decryption Message authentication Related SSPs: None
HMAC key (CSP)	112-256 bits	HMAC A3534, A3544, A3545, A3546, A3547, A4022, A4024, A4459	N/A	MD/EE Import: API input parameters From: Operator calling application (TOEPP) To: Cryptographic module Export: None	N/A	RAM	EVP_MAC_CTX_free	Use: Message authentication Related SSPs: None
Key-derivation key (CSP)	112-256 bits	KBKDF A3553	N/A	MD/EE Import: API input parameters From: Operator calling application (TOEPP) To: Cryptographic module	N/A	RAM	EVP_KDF_CTX_free	Use: KBKDF Key derivation Related SSPs: KBKDF Derived key

				Export: None				
DH Shared secret (CSP)	112-256 bits	KAS-FFC-SSC A3554 KDA OneStep HKDF ANS X9.42 KDF ANS X9.63 KDF SSH KDF TLS 1.2 KDF TLS 1.3 KDF A3525, A3526, A3530, A3531, A3532, A3533, A3534, A3544, A3545, A3546, A3547, A3553, A4022, A4023, A4024, A4459, A4460	N/A	MD/EE Import: API input parameters From: Operator calling application (TOEPP) To: Cryptographic module Export: API output parameters From: Cryptographic module To: Operator calling application (TOEPP)	SP 800-56Ar3 (DH shared secret computation)	RAM	EVP_KDF_CTX_free	Use: Shared secret computation KDA OneStep Key derivation HKDF Key derivation ANS X9.42 KDF Key derivation ANS X9.63 KDF Key derivation SSH KDF Key derivation TLS 1.2 KDF Key derivation TLS 1.3 KDF Key derivation Related SSPs: KDA OneStep Derived key HKDF Derived key ANS X9.42 KDF Derived key ANS X9.63 KDF Derived key SSH KDF Derived key TLS 1.2 KDF Derived key TLS 1.3 KDF Derived key DH private key DH public key
ECDH Shared secret (CSP)	112-256 bits	KAS-ECC-SSC KDA OneStep HKDF ANS X9.42 KDF ANS X9.63 KDF SSH KDF TLS 1.2 KDF TLS 1.3 KDF A3525, A3526, A3530, A3531, A3532, A3533, A3534, A3544, A3545, A3546, A3547, A3553, A4022, A4023, A4024, A4459, A4460	N/A	MD/EE Import: API input parameters From: Operator calling application (TOEPP) To: Cryptographic module Export: API output parameters From: Cryptographic module To: Operator	SP 800-56Ar3 (ECDH shared secret computation)	RAM	EVP_KDF_CTX_free	Use: Shared secret computation KDA OneStep Key derivation HKDF Key derivation ANS X9.42 KDF Key derivation ANS X9.63 KDF Key derivation SSH KDF Key derivation TLS 1.2 KDF Key derivation TLS 1.3 KDF Key derivation Related SSPs: KDA OneStep Derived key HKDF Derived

				calling application (TOEPP)				key ANS X9.42 KDF Derived key ANS X9.63 KDF Derived key SSH KDF Derived key TLS 1.2 KDF Derived key TLS 1.3 KDF Derived key EC private key EC public key
Password (CSP)	Password strength: 10^8 - 10^{128}	PBKDF2 A3534, A3544, A3545, A3546, A3547, A4022, A4024, A4459	N/A	MD/EE Import: API input parameters From: Operator calling application (TOEPP) To: Cryptographic module Export: None	N/A	RAM	EVP_KDF_CTX_free	Use: Password-based key derivation Related SSPs: PBKDF2 Derived key
KBKDF Derived key (CSP)	112-256 bits	KBKDF A3553	SP 800-133r2, Section 6.2	MD/EE Import: None Export: API output parameters From: Cryptographic module To: Operator calling application (TOEPP)	N/A	RAM	EVP_KDF_CTX_free	Use: KBKDF Key derivation Related SSPs: Key-derivation key
KDA OneStep Derived key (CSP)		KDA OneStep A3525						Use: KDA OneStep Key derivation Related SSPs: DH Shared secret ECDH Shared secret
HKDF Derived key (CSP)		HKDF A3526						Use: HKDF Key derivation Related SSPs: DH Shared secret ECDH Shared secret
ANS X9.42 KDF Derived key (CSP)		ANS X9.42 KDF A3534 A3544 A3545 A3546 A3547 A4022 A4024 A4459						Use: ANS X9.42 KDF Key derivation Related SSPs: DH Shared secret

								ECDH Shared secret
ANS X9.63 KDF Derived key (CSP)		ANS X9.63 KDF A3534 A3544 A3545 A3546 A3547 A4022 A4024 A4459						Use: ANS X9.63 KDF Key derivation Related SSPs: DH Shared secret ECDH Shared secret
SSH KDF Derived key (CSP)		SSH KDF A3530 A3531 A3532 A3533 A4023 A4460						Use: SSH KDF Key derivation Related SSPs: DH Shared secret ECDH Shared secret
TLS 1.2 KDF Derived key (CSP)		TLS 1.2 KDF A3544 A3545 A3546 A3547 A4022 A4459						Use: TLS 1.2 KDF Key derivation Related SSPs: DH Shared secret ECDH Shared secret
TLS 1.3 KDF Derived key (CSP)		TLS 1.3 KDF A3526						Use: TLS 1.3 KDF Key derivation Related SSPs: DH Shared secret ECDH Shared secret
PBKDF2 Derived key (CSP)		PBKDF2 A3534 A3544 A3545 A3546 A3547 A4022 A4024 A4459						Use: Password-based key derivation Related SSPs: Password
Entropy input (CSP)	112-336 bits	CTR_DRBG Hash_DRBG HMAC_DRBG A3570	N/A	Import: None Export: None	N/A	RAM	EVP_RAND_ CTX_free	Use: Random number generation Related SSPs: DRBG seed
DRBG seed (CSP) IG D.L compliant	CTR_DRBG: 128, 192, 256 bits Hash_DRBG: 128, 256 bits HMAC_DRBG: 128, 256 bits		CTR_DRBG Hash_DRBG HMAC_DRBG	Import: None Export: None	N/A	RAM	EVP_RAND_ CTX_free	Use: Random number generation Related SSPs: Entropy input Internal state (V, Key) Internal state (V, C)
Internal state (V, Key) (CSP) IG D.L		CTR_DRBG HMAC_DRBG A3570	CTR_DRBG HMAC_DRBG	Import: None Export: None	N/A	RAM	EVP_RAND_ CTX_free	Use: Random number generation

compliant								Related SSPs: DRBG seed
Internal state (V, C) (CSP) IG D.L compliant		Hash_DRBG A3570	Hash_DRBG					
DH private key (CSP)	112-200 bits	KAS-FFC-SSC A3554	SP 800-56Ar3 (safe primes) Section 5.6.1.1.4 Testing Candidates	MD/EE Import: API input parameters From: Operator calling application (TOEPP) To: Cryptographic module Export: API output parameters From: Cryptographic module To: Operator calling application (TOEPP)	N/A	RAM	EVP_PKEY_free	Use: Shared secret computation Key pair generation Key pair verification Related SSPs: DH public key Intermediate key generation value
DH public key (PSP)	112-200 bits							Use: Shared secret computation Key pair generation Key pair verification Related SSPs: DH private key Intermediate key generation value
EC private key (CSP)	112, 128, 192, 256 bits	KAS-ECC-SSC ECDSA A3534, A3544, A3545, A3546, A3547, A4022, A4024, A4459	FIPS 186-4 Appendix B.4.2 Testing Candidates	MD/EE Import: API input parameters From: Operator calling application (TOEPP) To: Cryptographic module Export: API output parameters From: Cryptographic module To: Operator calling application (TOEPP)	N/A	RAM	EVP_PKEY_free	Use: Shared secret computation Signature generation Key pair generation Key pair verification Related SSPs: EC public key Intermediate key generation value
EC public key (PSP)	112, 128, 192, 256 bits							Use: Shared secret computation Signature verification Key pair generation Key pair verification Related SSPs: EC private key Intermediate key generation

								value
RSA private key (CSP)	112-256 bits	RSA A3544, A3545 A3546, A3547, A4022, A4459	FIPS 186-4 Appendix B.3.6 Probable Primes with Conditions Based on Auxiliary Probable Primes	MD/EE Import: API input parameters From: Operator calling application (TOEPP) To: Cryptographic module Export: API output parameters From: Cryptographic module To: Operator calling application (TOEPP)	N/A	RAM	EVP_PKEY_free	Use: Key pair generation Signature generation Related SSPs: RSA public key Intermediate key generation value
RSA public key (PSP)	80-256 bits							Use: Key pair generation Signature verification Related SSPs: RSA private key Intermediate key generation value
Intermediate key generation value (CSP)	112-256 bits	CKG vendor affirmed	SP 800-133r2 Section 4, 5.1, and 5.2	Import: None Export: None	N/A	RAM	Automatic	Use: Key pair generation Related SSPs: DH private key DH public key EC private key EC public key RSA private key RSA public key

Table 11 - SSPs

9.1 Random bit generators

The module employs two Deterministic Random Bit Generator (DRBG) implementations based on SP 800-90Ar1. These DRBGs are used internally by the module (e.g. to generate seeds for asymmetric key pairs and random numbers for security functions). They can also be accessed using the specified API functions. The following parameters are used:

1. Private DRBG: AES-256 CTR_DRBG with derivation function. This DRBG is used to generate secret random values (e.g., during asymmetric key pair generation). It can be accessed using `RAND_priv_bytes`.
2. Public DRBG: AES-256 CTR_DRBG with derivation function. This DRBG is used to generate general purpose random values that do not need to remain secret (e.g. initialization vectors). It can be accessed using `RAND_bytes`.

These DRBGs will always employ prediction resistance. More information regarding the configuration and design of these DRBGs can be found in the module's manual pages.

Entropy Source	Minimum number	Details

	of bits of entropy	
SP 800-90B compliant Non-Physical Entropy Source (ESV cert. E48)	238 bits of entropy in the 256-bit output	OpenSSL CPU Jitter 2.2.0 entropy source is located within the physical perimeter of the module but partially outside the cryptographic boundary of the module.

Table 12 - Non-Deterministic Random Number Generation Specification

The module generates SSPs (e.g., keys) whose strengths are modified by available entropy.

9.2 SSP generation

The module implements Cryptographic Key Generation (CKG, vendor affirmed), compliant with SP 800-133r2. When random values are required, they are obtained from the SP 800-90Ar1 approved DRBG, compliant with Section 4 of SP 800-133r2. The following methods are implemented:

- Safe primes key pair generation: compliant with SP 800-133r2, Section 5.2, which maps to SP 800-56Ar3. The method described in Section 5.6.1.1.4 of SP 800-56Ar3 ("Testing Candidates") is used.
- RSA key pair generation: compliant with SP 800-133r2, Section 5.1, which maps to FIPS 186-4. The method described in Appendix B.3.6 of FIPS 186-4 ("Probable Primes with Conditions Based on Auxiliary Probable Primes") is used.
- ECC (ECDH and ECDSA) key pair generation: compliant with SP 800-133r2, Section 5.1, which maps to FIPS 186-4. The method described in Appendix B.4.2 of FIPS 186-4 ("Testing Candidates") is used.

Additionally, the module implements the following key derivation methods:

- KBKDF: compliant with SP 800-108r1. This implementation can be used to generate secret keys from a pre-existing key-derivation-key.
- KDA OneStep, HKDF: compliant with SP 800-56Cr2. These implementations shall only be used to generate secret keys in the context of an SP 800-56Ar3 key agreement scheme.
- ANS X9.42 KDF, ANS X9.63 KDF: compliant with SP 800-135r1. These implementations shall only be used to generate secret keys in the context of an ANS X9.42-2001 resp. ANS X9.63-2001 key agreement scheme.
- SSH KDF, TLS 1.2 KDF, TLS 1.3 KDF: compliant with SP 800-135r1. These implementations shall only be used to generate secret keys in the context of the SSH, TLS 1.2, or TLS 1.3 protocols, respectively.
- PBKDF2: compliant with option 1a of SP 800-132. This implementation shall only be used to derive keys for use in storage applications.

Intermediate key generation values are not output from the module and are explicitly zeroized after processing the service.

9.3 SSP establishment

The module provides Diffie-Hellman (DH) and Elliptic Curve Diffie-Hellman (ECDH) shared secret computation compliant with SP800-56Ar3, in accordance with scenario 2 (1) of FIPS 140-3 IG D.F.

For Diffie-Hellman, the module supports the use of the safe primes defined in RFC 3526 (IKE) and RFC 7919 (TLS). Note that the module only implements key pair generation, key pair verification, and shared secret computation. No other part of the IKE or TLS protocols is implemented (with the exception of the TLS 1.2 and 1.3 KDFs):

- IKE (RFC 3526):
    - MODP-2048 (ID = 14)
    - MODP-3072 (ID = 15)
    - MODP-4096 (ID = 16)
    - MODP-6144 (ID = 17)
    - MODP-8192 (ID = 18)
- TLS (RFC 7919)
    - ffdhe2048 (ID = 256)
    - ffdhe3072 (ID = 257)
    - ffdhe4096 (ID = 258)
    - ffdhe6144 (ID = 259)
    - ffdhe8192 (ID = 260)

For Elliptic Curve Diffie-Hellman, the module supports the NIST-defined P-224, P-256, P-384, and P-521 curves.

According to FIPS 140-3 IG D.B, the key sizes of DH and ECDH shared secret computation provide 112-200 resp. 112-256 bits of security strength in an approved mode of operation.

SP 800-56Ar3 assurances:

To comply with the assurances found in Section 5.6.2 of SP 800-56Ar3, the operator must use the module together with an application that implements the TLS protocol. Additionally, the module's approved "Key pair generation" service must be used to generate ephemeral Diffie- Hellman or EC Diffie-Hellman key pairs, or the key pairs must be obtained from another FIPS-validated module. As part of this service, the module will internally perform the full public key validation of the generated public key. The module's shared secret computation service will internally perform the full public key validation of the peer public key, complying with Sections 5.6.2.2.1 and 5.6.2.2.2 of SP 800-56Ar3.

The module also supports the AES KW and AES KWP key wrapping mechanisms. These algorithms can be used to wrap SSPs with a security strength of 128, 192, or 256 bits, depending on the wrapping key size.

9.4 SSP entry/output

The module only supports SSP entry and output to and from the calling application running on the same operational environment. This corresponds to manual distribution, electronic entry/output ("CM Software to/from App via TOEPP Path") per FIPS 140-3 IG 9.5.A Table 1. There is no entry or output of cryptographically protected SSPs.

SSPs can be entered into the module via API input parameters, when required by a service. SSPs can also be output from the module via API output parameters, immediately after generation of the SSP (see Section 9.2).

9.5 SSP storage

SSPs are provided to the module by the calling application and are destroyed when released by the appropriate API function calls. The module does not perform persistent storage of SSPs.

9.6 SSP zeroization

The memory occupied by SSPs is allocated by regular memory allocation operating system calls. The operator application is responsible for calling the appropriate destruction functions provided in the module's API. The destruction functions (listed in Table 11) overwrite the memory occupied by SSPs with zeroes and de-allocate the memory with the regular memory de-allocation operating system call. All data output is inhibited during zeroization.

10 Self-tests

The module performs pre-operational self-tests and conditional self-tests. While the module is executing the self-tests, services are not available, and data output (via the data output interface) is inhibited until the tests are successfully completed. The module does not return control to the calling application until the tests are completed.

Both conditional and pre-operational self-tests can be executed on-demand by unloading and subsequently re-initializing the module.

All the self-tests are listed in Table 12, with the respective condition under which those tests are performed. Note that the pre-operational integrity test is only executed after all cryptographic algorithm self-tests (CASTs) executed successfully.

Algorithm	Parameters	Condition	Type	Test
HMAC	SHA-256	Initialization (after CASTs)	Pre-operational Integrity Test	MAC tag verification on fips.so file
SHA-1	N/A	Initialization	Cryptographic Algorithm Self-Test	KAT digest generation
SHA-512	N/A	Initialization	Cryptographic Algorithm Self-Test	KAT digest generation
SHA3-256	N/A	Initialization	Cryptographic Algorithm Self-Test	KAT digest generation
AES GCM	256-bit key	Initialization	Cryptographic Algorithm Self-Test	KAT encryption and decryption
AES ECB	128-bit key	Initialization	Cryptographic Algorithm Self-Test	KAT decryption
KBKDF	HMAC SHA-256 in counter mode	Initialization	Cryptographic Algorithm Self-Test	KAT key derivation
KDA OneStep	SHA-224	Initialization	Cryptographic Algorithm Self-Test	KAT key derivation
HKDF	SHA-256	Initialization	Cryptographic Algorithm Self-Test	KAT key derivation
ANS X9.42 KDF	AES-128 KW with SHA-1	Initialization	Cryptographic Algorithm Self-Test	KAT key derivation
ANS X9.63 KDF	SHA-256	Initialization	Cryptographic Algorithm Self-Test	KAT key derivation
SSH KDF	SHA-1	Initialization	Cryptographic Algorithm Self-Test	KAT key derivation
TLS 1.2 KDF	SHA-256	Initialization	Cryptographic Algorithm Self-Test	KAT key derivation
TLS 1.3 KDF	SHA-256	Initialization	Cryptographic Algorithm Self-Test	KAT key derivation
PBKDF2	SHA-256 with 4096 iterations	Initialization	Cryptographic Algorithm	KAT password-based key derivation

Algorithm	Parameters	Condition	Type	Test
	and 288-bit salt		Self-Test	
CTR_DRBG	AES-128 with derivation function and prediction resistance	Initialization	Cryptographic Algorithm Self-Test	KAT DRBG generation and reseed
Hash_DRBG	SHA-256 with prediction resistance	Initialization	Cryptographic Algorithm Self-Test	KAT DRBG generation and reseed
HMAC_DRBG	SHA-1 with prediction resistance	Initialization	Cryptographic Algorithm Self-Test	KAT DRBG generation and reseed
KAS-FFC-SSC	ffdhe2048	Initialization	Cryptographic Algorithm Self-Test	KAT shared secret computation
KAS-ECC-SSC	P-256	Initialization	Cryptographic Algorithm Self-Test	KAT shared secret computation
RSA	PKCS#1 v1.5 with SHA-256 and 2048-bit key	Initialization	Cryptographic Algorithm Self-Test	KAT signature generation and verification
ECDSA	SHA-256 and P-224, P-256, P-384, and P-521	Initialization	Cryptographic Algorithm Self-Test	KAT signature generation and verification
DH	N/A	DH key pair generation	Pair-wise Consistency Test	Section 5.6.2.1.4 pair-wise consistency
RSA	PKCS#1 v1.5 with SHA-256	RSA key pair generation	Pair-wise Consistency Test	Sign/verify pair-wise consistency
ECDSA	SHA-256	EC key pair generation	Pair-wise Consistency Test	Sign/verify pair-wise consistency

Table 13 - Self-Tests

10.1 Pre-operational tests

The module performs pre-operational tests automatically when the module is powered on. The pre-operational self-tests ensure that the module is not corrupted. The module transitions to the operational state only after the pre-operational self-tests are passed successfully.

The types of pre-operational self-tests are described in the next sub-sections.

10.1.1 Pre-operational software integrity test

The integrity of the shared library component of the module is verified by comparing an HMAC SHA-256 value calculated at run time with the HMAC SHA-256 value embedded in the fips.so file that was computed at build time.

If the software integrity test fails, the module transitions to the error state (Section 10.3). As mentioned previously, the HMAC and SHA-256 algorithms go through their respective CASTs before the software integrity test is performed.

10.2 Conditional self-tests

10.2.1 Conditional cryptographic algorithm tests

The module performs self-tests on all approved cryptographic algorithms as part of the approved services supported in the approved mode of operation, using the tests shown in Table 13. Data output through the data output interface is inhibited during the self-tests. If any of these tests fails, the module transitions to the error state (Section 10.3).

10.2.2 Conditional pair-wise consistency test

Upon generation of a DH, RSA or EC key pair, the module will perform a pair-wise consistency test (PCT) as shown in Table 13, which provides some assurance that the generated key pair is well formed. For DH key pairs, this tests consists of the PCT described in Section 5.6.2.1.4 of SP 800-56Ar3. For RSA and EC key pairs, this test consists of a signature generation and a signature verification operation. If the test fails, the module transitions to the error state (Section 10.3).

10.3 Error states

If the module fails any of the self-tests, the module enters the error state. In the error state, the module immediately stops functioning and ends the application process. Consequently, the data output interface is inhibited, and the module accepts no more inputs or requests (as the module is no longer running).

Table 8 lists the error states and the status indicator values that explain the error that has occurred.

Error State	Cause of Error	Status Indicator
Error	Software integrity test failure	Module will not load
	CAST failure	Module will not load
	PCT failure	Module stops functioning

Table 14 - Error States

11 Life-cycle assurance

11.1 Delivery and operation

The module is distributed as a part of the Red Hat Enterprise Linux 9 (RHEL 9) package in the form of the openssl-3.0.1-46.el9_0.3 RPM package.

11.1.1 End of life procedures

As the module does not persistently store SSPs, secure sanitization of the module consists of unloading the module. This will zeroize all SSPs in volatile memory. Then, if desired, the openssl-3.0.1-46.el9_0.3 RPM package can be uninstalled from the RHEL 9 system.

11.2 Crypto Officer guidance

Before the openssl-3.0.1-46.el9_0.3 RPM package is installed, the RHEL 9 system must operate in the approved mode. This can be achieved by:

- Adding the `fips=1` option to the kernel command line during the system installation. During the software selection stage, do not install any third-party software. More information can be found at the vendor documentation.
- Switching the system into the approved mode after the installation. Execute the `fips-mode-setup --enable` command. Restart the system. More information can be found at the vendor documentation.

In both cases, the Crypto Officer must verify the RHEL 9 system operates in the approved mode by executing the `fips-mode-setup --check` command, which should output “FIPS mode is enabled.”

After installation of the openssl-3.0.1-46.el9_0.3 RPM package, the Crypto Officer must execute the `openssl list -providers` command. The Crypto Officer must ensure that the `fips` provider is listed in the output as follows:

```
fips
  name: Red Hat Enterprise Linux 9 - OpenSSL FIPS Provider
  version: 3.0.1-3f45e68ee408cd9c
  status: active
```

The cryptographic boundary consists only of the FIPS provider as listed. If any other OpenSSL or third-party provider is invoked, the user is not interacting with the module specified in this Security Policy.

11.2.1 AES GCM IV

The Crypto Officer shall consider the following requirements and restrictions when using the module.

For TLS 1.2, the module offers the AES GCM implementation and uses the context of Scenario 1 of FIPS 140-3 IG C.H. OpenSSL 3 is compliant with SP 800-52r2 Section 3.3.1 and the mechanism for IV generation is compliant with RFC 5288 and 8446.

The module does not implement the TLS protocol. The module’s implementation of AES GCM is used together with an application that runs outside the module’s cryptographic boundary. The design of the TLS protocol implicitly ensures that the counter (the nonce_explicit part of the IV) does not exhaust the maximum number of possible values for a given session key.

In the event the module's power is lost and restored, the consuming application must ensure that a new key for use with the AES GCM key encryption or decryption under this scenario shall be established.

Alternatively, the Crypto Officer can use the module's API to perform AES GCM encryption using internal IV generation. These IVs are always 96 bits and generated using the approved DRBG internal to the module's boundary.

The module also provides a non-approved AES GCM encryption service which accepts arbitrary external IVs from the operator. This service can be requested by invoking the `EVP_EncryptInit_ex2` API function with a non-NULL `iv` value. When this is the case, the API will set a non-approved service indicator as described in Section 4.3.

Finally, for TLS 1.3, the AES GCM implementation uses the context of Scenario 5 of FIPS 140-3 IG C.H. The protocol that provides this compliance is TLS 1.3, defined in RFC8446 of August 2018, using the cipher-suites that explicitly select AES GCM as the encryption/decryption cipher (Appendix B.4 of RFC8446). The module supports acceptable AES GCM cipher suites from Section 3.3.1 of SP800-52r2. TLS 1.3 employs separate 64-bit sequence numbers, one for protocol records that are received, and one for protocol records that are sent to a peer. These sequence numbers are set at zero at the beginning of a TLS 1.3 connection and each time when the AES-GCM key is changed. After reading or writing a record, the respective sequence number is incremented by one. The protocol specification determines that the sequence number should not wrap, and if this condition is observed, then the protocol implementation must either trigger a re-key of the session (i.e., a new key for AES-GCM), or terminate the connection.

11.2.2 AES XTS

In compliance with IG C.I, the module implements the check to ensure that the two AES keys used in AES XTS are not identical.

The length of a single data unit encrypted or decrypted with AES XTS shall not exceed 2^{20} AES blocks, that is 16MB, of data per XTS instance. An XTS instance is defined in Section 4 of SP 800-38E.

The XTS mode shall only be used for the cryptographic protection of data on storage devices. It shall not be used for other purposes, such as the encryption of data in transit.

11.2.3 Key derivation using SP 800-132 PBKDF2

The module provides password-based key derivation (PBKDF2), compliant with SP 800-132. The module supports option 1a from Section 5.4 of SP 800-132, in which the Master Key (MK) or a segment of it is used directly as the Data Protection Key (DPK). In accordance to SP 800-132 and FIPS 140-3 IG D.N, the following requirements shall be met:

- Derived keys shall only be used in storage applications. The MK shall not be used for other purposes. The module accepts a minimum length of 112 bits for the MK or DPK.
- Passwords or passphrases, used as an input for the PBKDF2, shall not be used as cryptographic keys.
- The minimum length of the password or passphrase accepted by the module is 8 characters. This will result in a password strength equal to 10^8. Combined with the minimum iteration count as described below, this provides an acceptable trade-off between user experience and security against brute-force attacks.
- A portion of the salt, with a length of at least 128 bits, shall be generated randomly using the SP 800-90Ar1 DRBG provided by the module.

- The iteration count shall be selected as large as possible, as long as the time required to generate the key using the entered password is acceptable for the users. The module only allows minimum iteration count to be 1000.

12 Mitigation of other attacks

Certain cryptographic subroutines and algorithms are vulnerable to timing analysis. The module mitigates this vulnerability by using constant-time implementations. This includes, but is not limited to:

- Big number operations: computing GCDs, modular inversion, multiplication, division, and modular exponentiation (using Montgomery multiplication)
- Elliptic curve point arithmetic: addition and multiplication (using the Montgomery ladder)
- Vector-based AES implementations

In addition, RSA, ECDSA, ECDH, and DH employ blinding techniques to further impede timing and power analysis. No configuration is needed to enable the aforementioned countermeasures.

Appendix A. Glossary and abbreviations

AES	**Advanced Encryption Standard**
AES-NI	**Advanced Encryption Standard New Instructions**
API	**Application Programming Interface**
CAST	**Cryptographic Algorithm Self-Test**
CAVP	**Cryptographic Algorithm Validation Program**
CBC	**Cipher Block Chaining**
CCM	**Counter with Cipher Block Chaining-Message Authentication Code**
CFB	**Cipher Feedback**
CKG	**Cryptographic Key Generation**
CMAC	**Cipher-based Message Authentication Code**
CMVP	**Cryptographic Module Validation Program**
CPACF	**CP Assist for Cryptographic Functions**
CSP	**Critical Security Parameter**
CTR	**Counter**
CTS	**Ciphertext Stealing**
DH	**Diffie-Hellman**
DRBG	**Deterministic Random Bit Generator**
ECB	**Electronic Code Book**
ECC	**Elliptic Curve Cryptography**
ECDH	**Elliptic Curve Diffie-Hellman**
ECDSA	**Elliptic Curve Digital Signature Algorithm**
EVP	**Envelope**
FFC	**Finite Field Cryptography**
FIPS	**Federal Information Processing Standards**
GCM	**Galois Counter Mode**
GMAC	**Galois Counter Mode Message Authentication Code**
HKDF	**HMAC-based Key Derivation Function**
HMAC	**Keyed-Hash Message Authentication Code**
IKE	**Internet Key Exchange**
KAS	**Key Agreement Scheme**
KAT	**Known Answer Test**
KBKDF	**Key-based Key Derivation Function**
KW	**Key Wrap**
KWP	**Key Wrap with Padding**
MAC	**Message Authentication Code**
NIST	**National Institute of Science and Technology**
OAEP	**Optimal Asymmetric Encryption Padding**

OFB	**Output Feedback**
PAA	**Processor Algorithm Acceleration**
PCT	**Pair-wise Consistency Test**
PBKDF2	**Password-based Key Derivation Function v2**
PKCS	**Public-Key Cryptography Standards**
PSS	**Probabilistic Signature Scheme**
RSA	**Rivest, Shamir, Addleman**
SHA	**Secure Hash Algorithm**
SSC	**Shared Secret Computation**
SSH	**Secure Shell**
SSP	**Sensitive Security Parameter**
TLS	**Transport Layer Security**
XOF	**Extendable Output Function**
XTS	**XEX-based Tweaked-codebook mode with cipher text Stealing**

Appendix B. References

ANS X9.42-2001	**Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography** **2001** https://webstore.ansi.org/standards/ascx9/ansix9422001
ANS X9.63-2001	**Public Key Cryptography for the Financial Services Industry, Key Agreement and Key Transport Using Elliptic Curve Cryptography** **2001** https://webstore.ansi.org/standards/ascx9/ansix9632001
FIPS 140-3	**FIPS PUB 140-3 - Security Requirements For Cryptographic Modules** **March 2019** https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf
FIPS 140-3 IG	**Implementation Guidance for FIPS PUB 140-3 and the Cryptographic Module Validation Program** https://csrc.nist.gov/Projects/cryptographic-module-validation-program/fips-140-3-ig-announcements
FIPS 180-4	**Secure Hash Standard (SHS)** **March 2012** https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
FIPS 186-4	**Digital Signature Standard (DSS)** **July 2013** https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS 197	**Advanced Encryption Standard** **November 2001** https://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
FIPS 198-1	**The Keyed Hash Message Authentication Code (HMAC)** **July 2008** https://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
FIPS 202	**SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions** **August 2015** https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
PKCS#1	**Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1** **February 2003** https://www.ietf.org/rfc/rfc3447.txt
RFC 3526	**More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)** **May 2003** https://www.ietf.org/rfc/rfc3526.txt
RFC 5288	**AES Galois Counter Mode (GCM) Cipher Suites for TLS** **August 2008** https://www.ietf.org/rfc/rfc5288.txt

RFC 7919	**Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)** **August 2016** https://www.ietf.org/rfc/rfc7919.txt
RFC 8446	**The Transport Layer Security (TLS) Protocol Version 1.3** **August 2018** https://www.ietf.org/rfc/rfc8446.txt
SP 800-38A	**Recommendation for Block Cipher Modes of Operation Methods and Techniques** **December 2001** https://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
SP 800-38A Addendum	**Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode** **October 2010** https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a-add.pdf
SP 800-38B	**Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication** **May 2005** https://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf
SP 800-38C	**Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality** **May 2004** https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf
SP 800-38D	**Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC** **November 2007** **https://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf**
SP 800-38E	**Recommendation for Block Cipher Modes of Operation: The XTS AES Mode for Confidentiality on Storage Devices** **January 2010** https://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf
SP 800-38F	**Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping** **December 2012** https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf
SP 800-52r2	**Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations** **August 2019** https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf
SP 800-56Ar3	**Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography** **April 2018** https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf
SP 800-56Cr2	**Recommendation for Key-Derivation Methods in Key-Establishment Schemes** **August 2020** https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr2.pdf

SP 800-90Ar1 **Recommendation for Random Number Generation Using Deterministic Random Bit Generators**
June 2015
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

SP 800-90B **Recommendation for the Entropy Sources Used for Random Bit Generation**
January 2018
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf

SP 800-108r1 **NIST Special Publication 800-108 - Recommendation for Key Derivation Using Pseudorandom Functions**
August 2022
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-108r1.pdf

SP 800-131Ar2 **Transitioning the Use of Cryptographic Algorithms and Key Lengths**
March 2019
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf

SP 800-132 **Recommendation for Password-Based Key Derivation - Part 1: Storage Applications**
December 2010
https://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf

SP 800-133r2 **Recommendation for Cryptographic Key Generation**
June 2020
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-133r2.pdf

SP 800-135r1 **Recommendation for Existing Application-Specific Key Derivation Functions**
December 2011
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf

SP 800-140B **CMVP Security Policy Requirements**
March 2020
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-140B.pdf