ORACLE
Linux

FIPS 140-2 Non-Proprietary Security Policy

Oracle Linux 8 GnuTLS Cryptographic Module

FIPS 140-2 Level 1 Validation

Software Version: R8-8.4.0

Date: July 6th, 2022

Title: Oracle Linux 8 Gnu TLS Cryptographic Module Security Policy
Date: July 6th, 2022
Author: Oracle Security Evaluations – Global Product Security
Contributing Authors:
atsec informatioin security
Oracle Linux Engineering

Oracle Corporation
World Headquarters
2300 Oracle Way
Austin, TX 78741
U.S.A.
Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

ORACLE

TABLE OF CONTENTS

List of Tables

List of Figures

1. Introduction

1.1 Overview

This document is the Security Policy for the Oracle Linux 8 GnuTLS Cryptographic Module by Oracle Corporation. Oracle Linux 8 GnuTLS Cryptographic Module is also referred to as “the Module or Module”. This Security Policy specifies the security rules under which the module shall operate to meet the requirements of FIPS 140-2 Level 1. It also describes how the Oracle Linux 8 GnuTLS Cryptographic Module functions in order to meet the FIPS requirements, and the actions that operators must take to maintain the security of the module.

This Security Policy describes the features and design of the Oracle Linux 8 GnuTLS Cryptographic Module using the terminology contained in the FIPS 140-2 specification. FIPS 140-2, Security Requirements for Cryptographic Module specifies the security requirements that will be satisfied by a cryptographic module utilized within a security system protecting sensitive but unclassified information. The NIST/CSE Cryptographic Module Validation Program (CMVP) validates cryptographic module to FIPS 140-2. Validated products are accepted by the Federal agencies of both the USA and Canada for the protection of sensitive or designated information.

1.2 Document Organization

The FIPS 140-2 Submission Package contains:

- Oracle Linux 8 GnuTLS Cryptographic Module Non-Proprietary Security Policy
- Other supporting documentation as additional references

With the exception of this Non-Proprietary Security Policy, the FIPS 140-2 Validation Documentation is proprietary to Oracle and is releasable only under appropriate non-disclosure agreements. For access to these documents, please contact Oracle.

2. Oracle Linux 8 GnuTLS Cryptographic Module

2.1 Functional Overview

The Oracle Linux 8 Gnu TLS Cryptographic Module is a set of libraries implementing general purpose cryptographic algorithms and network protocols. The module supports the Transport Layer Security (TLS) Protocol defined in [RFC 5246] and the Datagram Transport Layer Security (DTLS) Protocol defined in [RFC 4347]. The module provides a C language Application Program Interface (API) for use by other calling applications that require cryptographic functionality or TLS/DTLS network protocols.

2.2 FIPS 140-2 Validation Scope

The following table shows the security level for each of the eleven sections of the validation.

Security Requirements Section	Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles and Services and Authentication	1
Finite State Machine Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	3
Mitigation of Other Attacks	1

Table 1: FIPS 140-2 Security Requirements

3. Cryptographic Module Specification

3.1 Definition of the Cryptographic Module

The Oracle Linux 8 GnuTLS Cryptographic Module is defined as a software-only multi-chip standalone module as defined by the requirements within FIPS PUB 140-2. The logical cryptographic boundary of the module consists of shared library files and their integrity check HMAC files, which are delivered through the Package Manager (RPM) as listed below:

Component	Description
libgnutls	This library provides the main interface which allows the calling applications to request cryptographic services. The Approved cryptographic algorithm implementations provided by this library include the TLS protocol, DRBG, RSA Key Generation, Diffie-Hellman and EC Diffie-Hellman.
libnettle	This library provides the cryptographic algorithm implementations, including AES, Triple-DES, SHA, HMAC, RSA Digital Signature, DSA and ECDSA.
libhogweed	This library includes the primitives used by libgnutls and libnettle to support the asymmetric cryptographic operations.
libgmp	This library provides the big number arithmetic operations to support the asymmetric cryptographic operations.
.hmac	The .hmac files contain the HMAC-SHA-256 values of its associated library for integrity check during the power-up.

Table 2: Oracle Linux 8 Gnu TLS Cryptographic Components

The module's logical boundary is the shared library files and their integrity check HMAC files, which are delivered through Oracle Linux Yum Public server listed in section 10.1.

All components of the module will be in the RPM versions gnutls-3.6.14-8.0.1.el8.x86_64.rpm, gmp-6.1.2-10.el8.x86_64.rpm, nettle-3.4.1-4.el8_3.x86_64.rpm or gnutls-3.6.14-8.0.1.el8.aarch64.rpm, gmp-6.1.2-10.el8.aarch64.rpm, nettle-3.4.1-4.el8_3.aarch64.rpm. The binary files and the HMAC files within the module's logical boundary are listed below:

- libgnutls library:
  - /usr/lib64/libgnutls.so.30.28 (64 bits)
  - /usr/lib64/.libgnutls.so.30.28.hmac (64 bits)

- libnettle library:
  - /usr/lib64/libnettle.so.6.5 (64 bits)
  - /usr/lib64/.libnettle.so.6.5.hmac (64 bits)

- libhogweed library:
  - /usr/lib64/libhogweed.so.4.5 (64 bits)
  - /usr/lib64/.libhogweed.so.4.5.hmac (64 bits)

- libgmp library:
  - /usr/lib64/libgmp.so.10.3.2 (64 bits)
  - /usr/lib64/fipscheck/libgmp.so.10.3.2.hmac (64 bits)

Figure 1 shows the logical block diagram of the module executing in memory on the host system.

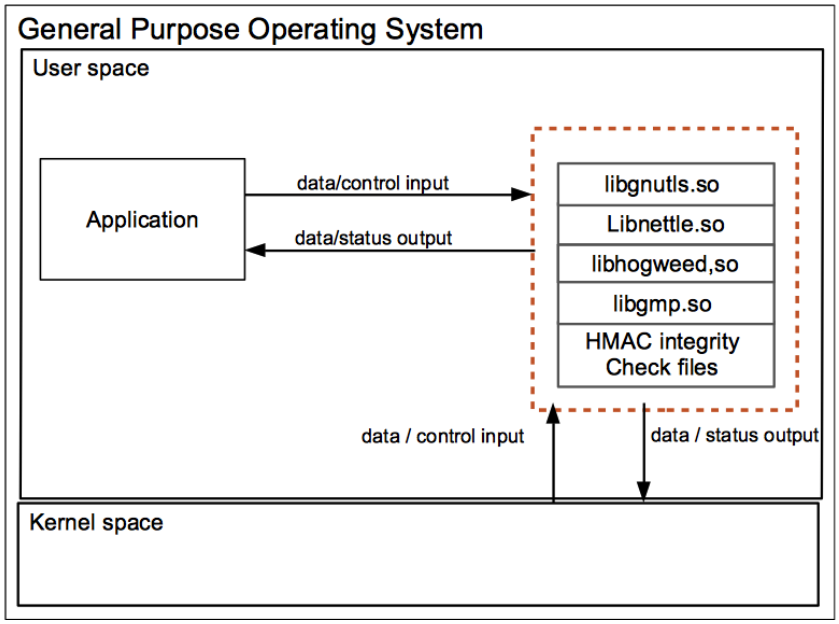

Figure 1: Oracle Linux 8 GnuTLS Logical Cryptographic Boundary

3.2 Definition of the Physical Cryptographic Boundary

The physical boundary of the module is the physical boundary of the test platform which is a General Purpose Computer (GPC). No components are excluded from the requirements of FIPS PUB 140-2. The following block diagram shows the hardware components of a GPC.

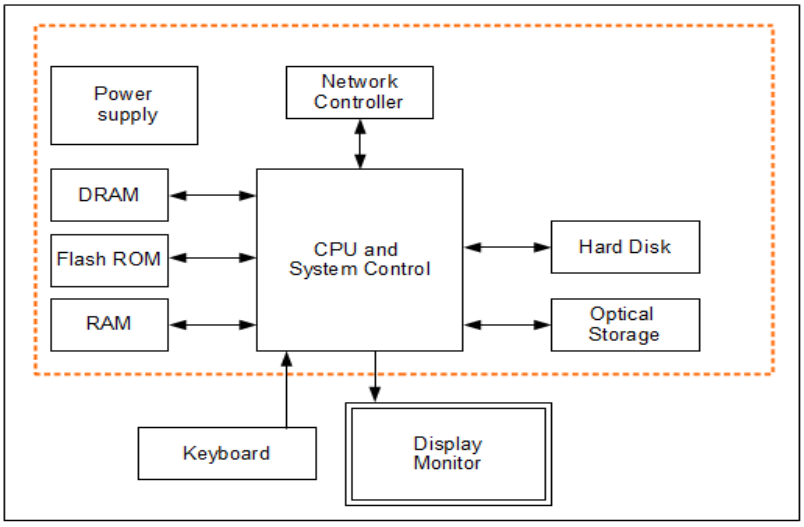

Figure 2: Oracle Linux 8 GnuTLS Hardware Block Diagram

3.3 Description of the Approved Modes of Operation

The module supports two modes of operation:

- In "FIPS mode" (the FIPS Approved mode of operation) only approved or allowed security functions with sufficient security strength can be used.
- In "non-FIPS mode" (the non-Approved mode of operation) only non-approved security functions can be used.

When the module is powered on, after the power-up self-tests are completed successfully, the module will be in FIPS Approved mode by default. Then the mode will be implicitly assumed depending on the services and security functions invoked.

3.4 Approved or Allowed Security Functions

The Oracle Linux 8 GnuTLS Cryptographic Module contains the following FIPS Approved Algorithms:

Approved or Allowed Security Functions		Certificate
Symmetric Algorithms		
AES	**AESNI:** AES in CBC, CCM, CMAC, and GCM modes (Key sizes 128 and 256 for all modes except CBC where key sizes are 128, 192, 256)	A 1704
	SSSE3 : AES in CBC, CMAC, and GCM modes (Key sizes 128 and 256 for all modes except CBC where key sizes are 128, 192, 256)	A 1705
	SSSE3_CFB8_CMAC: AES in CFB8 mode (Key sizes 128, 192, 256)	A 1707
	AESNI_CFB8_CMAC: AES in CFB8 mode (Key sizes 128, 192, 256)	A 1708
	Generic C: AES in CBC, CMAC, GCM and GMAC modes (Key sizes 128 and 256 for all modes except CBC where key sizes are 128, 192, 256)	A 1710
	Generic_C_XTS : AES in XTS mode (Key sizes 128 and 256)	A 1711
	C_CFB8 : AES in CFB8 mode (Key sizes 128, 192, 256)	A 1713
	CE AES in CBC, CCM, GCM modes (Key sizes 128 and 256 for all modes except CBC where key sizes are 128, 192, 256)	A 2560
Triple DES	**Generic_C:** **CBC** (KO 1, d/e)	A 1710
Cryptographic Key Generation (CKG)		
CKG	**NIST SP 800-133rev2** Asymmetric and Symmetric Cryptographic Key Generation	(Vendor Affirmed)
Secure Hash Standard (SHS)		
SHS	**SSSE3:** **SHA** (1, 224, 256, 384, 512)	A 1705
	Generic_C: **SHA** (1, 224, 256, 384, 512)	A 1710

Approved or Allowed Security Functions		Certificate
	CE **SHA** (1, 224, 256, 384, 512)	A 2560
SHA-3	**C_SHA-3:** SHA3-224, SHA3-256, SHA3-384, SHA3-512	A 1712
	SSSE3_SHA3: SHA3-224, SHA3-256, SHA3-384, SHA3-512	A 1706
Data Authentication Code		
HMAC	**SSSE3:** **HMAC-SHA** (1, 224, 256, 384, 512)	A 1705
	Generic_C: **HMAC-SHA** (1, 224, 256, 384, 512)	A 1710
	CE **HMAC-SHA** (1, 224, 256, 384, 512)	A 2560
Asymmetric Algorithms		
RSA	**Generic_C:** **FIPS186-4:** (Key Gen): Modulus Sizes: 2048, 3072, 4096 with hash size SHA2-384; PKCS 1.5 (Sig Gen): Modulus sizes 2048, 3072, 4096 with hash sizes SHA2-224, SHA2-256, SHA2-384, SHA2-512; PKCSPSS (Sig Gen): Modulus sizes 2048, 3072, 4096 with hash sizes SHA2-256, SHA2-384, SHA2-512; PKCS 1.5 (Sig Ver): Modulus sizes 2048, 3072, 4096 with hash sizes SHA2-224, SHA2-256, SHA2-384, SHA2-512; PKCSPSS (Sig Ver): Modulus sizes 2048, 3072, 4096 with hash sizes SHA2-256, SHA2-384, SHA2-512	A 1710
DSA	**Generic_C:** **FIPS186-4:** (Key Gen): L=2048, N=224; L=2048, N=256; L=3072, N=256; (PQG Gen, PQG Ver): L=2048, N=224; L=2048, N=256; L=3072, N=256; (with SHA2-384); (SIG Gen): L=2048, N=224 (with SHA2-224, SHA2-256, SHA2-384, SHA2-512); L=2048, N=256; L=3072, N=256 (with SHA2-256, SHA2-384, SHA2-512); (SIG Ver): L=2048, N=224 (with SHA1, SHA2-224, SHA2-256, SHA2-384, SHA2-512); L=2048, N=256 (with SHA1, SHA2-256, SHA2-384, SHA2-512); L=3072, N=256 (with SHA1, SHA2-256, SHA2-384, SHA2-512)	A 1710
ECDSA	**Generic_C:** **FIPS186-4:** (Key Gen, Key Ver): Curves P-256, P-384, P-521; (Sig Gen, Sig Ver): Curves P-256, P-384, P-521 with hash sizes SHA2-224, SHA2-256, SHA2-384, SHA2-512	A 1710
Random Number Generation NIST SP 900-90A		
DRBG	**Generic_C:** AES ctr DRBG (Key size 256); without Derivation Function, without Prediction Resistance and Reseeding implementation	A 1710
Key Agreement Scheme Using HMAC Key Derivation Function (SP 800-56Cr1)		
KDA HKDF SP	**TLS_v1.3:** Fixed Info Pattern: uPartyInfo || vPartyInfo	A 1709

Approved or Allowed Security Functions		Certificate
800-56Cr1	Fixed Info Encoding: concatenation Derived Key Length: 2048 Shared Secret Length: 224-65336 Increment 8 HMAC Algorithm: SHA2-224, SHA2-256, SHA2-384, SHA2-512	
Key Establishment (NIST SP 800-56Ar3)		
KAS-FFC-SSC SP 800-56Ar3	**Generic_C:** Domain Parameter Generation and Mod P Methods (ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192) Scheme: dhEphem KAS role: initiator, responder	A 1710
KAS-ECC-SSC SP 800-56Ar3	**Generic_C:** Domain Parameter Generation Methods (Curves P-256, P-384, P-521). Scheme: EphemeralUnified KAS role: initiator, responder	A 1710
Safe Primes Key Generation	**Generic_C:** Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	A 1710
KAS	KAS-FFC-SSC SP 800-56Ar3 with ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192; KAS-ECC-SSC SP 800-56Ar3 with curves P-256, P-384, P-521; KDF-TLS 1.0/1.1, 1.2 with SHA-256, SHA-384	A 1710
Key Derivation (NIST SP 800-135 Section 4.2 in TLS 1.0, 1.1, and 1.2)		
KDF-TLS (CVL)	TLS1.0/1.1, TLS 1.2 (SHA 256, 384)	A 1710
Password Based Key Derivation Function		
PBKDF2	**Generic_C:** HMAC Algorithm: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512 Password Length: 8-128 Increment 1 Salt Length: 128-4096 Increment 8 Key Data Length: 128-4096 Increment 8	A 1710
Key Transport Scheme (KTS)		
KTS	**AESNI:** AES-GCM Key Wrapping with 128 and 256 bit keys; AES-CCM Key Wrapping with 128 and 256 bit keys;	A 1704
	AES-CBC with 128 and 256 bit keys and HMAC-SHA1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512;	A 1704 (AES) A 1710
	SSSE3 : AES-GCM Key Wrapping with 128 and 256 bit keys; AES-CBC with 128 and 256 bit keys and HMAC-SHA1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512;	A 1705
	Generic C AES-GCM Key Wrapping with 128 and 256 bit keys; AES-CBC with 128 and 256 bit keys and HMAC-SHA1, HMAC-SHA2-224, HMAC-SHA2-256,	A 1710

Approved or Allowed Security Functions		Certificate
	HMAC-SHA2-384, HMAC-SHA2-512;	
	Triple-DES CBC with HMAC-SHA1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512;	A 1705 (HMAC) A 1710 A 2560 (HMAC)
	CE: AES-GCM modes Key Wrapping with 128 and 256 bit keys; AES-CCM modes Key Wrapping with 128 and 256 bit keys; AES-CBC with 128 and 256 bit keys and HMAC-SHA1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512;	A 2560
Entropy (NIST SP 800-90B)		
ENT(NP)	**NIST SP 800-90B**	N/A

Table 3: FIPS Approved or Allowed Security Functions

The module supports different AES and SHA implementations based on the underlying platform's capability. The module supports the use of AES-NI and SSSE3 when it is operated in an x86-64 architecture environment. When the AES-NI is enabled in the operating environment, the module performs the AES operations using the supports from the AES-NI instructions; when the AES-NI is disabled in the operating environment, the module performs the AES operations using the supports from the Supplemental Streaming SIMD Extensions 3 (SSSE3). The module also performs SHA operations using the supports from the SSSE3.

Additionally, the module supports ARM Cryptographic Extensions (CE) for AES and SHA operations when operated in an ARM architecture environment. The AES and SHA implementations that uses the AES-NI, SSSE3 and CE supports and their related algorithms have been CAVP tested and functional tested. Although the module implements different implementations for AES and SHA, only one implementation for one algorithm will ever be available for AES, SHA and HMAC cryptographic services at run-time.

3.5 Non-Approved but Allowed Security Functions

The following are considered non-Approved but allowed security functions:

Algorithm	Usage
RSA Key Wrapping	Key wrapping, size between 2048 and 16384 bits or more
MD5 (no security claimed per IG 1.23)	Used in TLS PRF only.

Table 4: Non-Approved but Allowed Security Functions

3.6 Non-Approved Security Functions

The following services are non-Approved and use of these algorithms will put the module in the non-approved mode of operation implicitly. The services associated with these algorithms are specified in section 7.3:

Algorithm	Usage
AES-SIV	Encrypt/Decrypt
Camellia	Encrypt/Decrypt
DES	Encrypt/Decrypt
Diffie-Hellman	Key Agreement using keys less than 2048 bits

Algorithm	Usage
	Keys generated with domain parameters other than safe primes
RSA	FIPS 186-2 Key Generation
	FIPS 186-4 Signature Generation/Verification, Key Generation with keys smaller than 2048 bits and larger than 4096 bits
	Key Wrapping with keys less than 2048 bits
DSA	Parameter/Key generation, Signature Generation/Verification with keys not listed in Table 3
	FIPS 186-4 Signature Verification with non-approved message digests
	FIPS 186-4 Signature Generation with non-approved message digests or SHA-1
ECDSA	Key Generation/Verification, Signature Generation/Verification with curves not listed in Table 3
	Key Generation with curves not listed in Table 3
RSA and ECDSA	FIPS 186-4 Signature Generation/Verification with non-approved message digests or SHA-1
GOST	GOST Hash R 34.11-94 (RFC4357)
MD2	Hashing
MD4	Hashing
MD5	Hashing
RC2	Encrypt/Decrypt
RC4	Encrypt/Decrypt
RIPEMD-160	Hashing
Salsa-20	Encrypt/Decrypt
SHA-1	Signature Generation
UMAC	Authenticated Data Integrity of a message
Streebog-256 and Streebog-512	Hashing
ChaCha20	Encrypt/Decrypt
Poly 1305	Authenticated Encryption/Decryption
Ed25519 curve	Digital Signature
EdDSA	Digital Signature

Table 5: Non-Approved Functions

4. Module Ports and Interfaces

As a software-only module, the module does not have physical ports. For the purpose of FIPS 140-2 validation, the physical ports of the module are interpreted to be the physical ports of the hardware platform on which it runs. The logical interface is a C-language Application Program Interface (API) through libgnutls library.

The Data Input interface consists of the input parameters of the API functions. The Data Output interface consists of the output parameters of the API functions. The Control Input interface consists of the actual API functions. The Status Output interface includes the return values of the API functions. The module can be accessed by utilizing the API it exposes. The table below, shows the mapping of ports and interfaces as per FIPS 140-2 Standard.

FIPS 140 Interface	Physical Port	Module Interfaces
Data Input	Ethernet Ports	API input parameters, kernel I/O – network or files on file system, TLS protocol
Data Output	Ethernet Ports	API output parameters, kernel I/O – network or files on file system, TLS protocol
Control Input	Management Ethernet Port, USB for Keyboard/Mouse, Serial Port	API function calls, TLS protocol
Status Output	Management Ethernet Port, Serial Port	API return codes, error message, TLS protocol

Table 6: Mapping of FIPS 140 Logical Interfaces to Logical Ports

Note: The module is an implementation to support the TLS protocol defined in [RFC5246] and TLS is a port networking interface to provide secure channel between entities. When the calling application sends the data to the module, the module packages the data according to the TLS standard and send to other entity confidentially and integrity. The module is considered as a user interface to use the TLS protocol to communicate with other remote entities securely through the network.

5. Physical Security

The Module is comprised of software only and thus does not claim any physical security.

6. Operational Environment

6.1 Tested Environments

The Module was tested on the following environments with and without PAA (i.e. AES-NI):

Operating Environment	Processor	Hardware
Oracle Linux 8.4 64-bit	Intel® Xeon® Platinum 8167M	Oracle Server X7-2C
Oracle Linux 8.4 64-bit	AMD EPYC™ 7551	Oracle Server E1-2C
Oracle Linux 8.4 64-bit	Ampere®Altra® Neoverse-N1	Oracle Server A1-2C

Table 7: Tested Operating Environment

6.2 Vendor Affirmed Environments

The following platforms have not been tested as part of the FIPS 140-2 level 1 certification however Oracle "vendor affirms" that these platforms are equivalent to the tested and validated platforms. Additionally, Oracle affirms that the module will function the same way and provide the same security services on any of the systems listed below.

Operating Environment	Hardware
Oracle Linux 8 64-bit	Oracle X Series Servers
Oracle Linux 8 64-bit	Oracle E Series Servers
Oracle Linux 8 64-bit	Oracle A Series Servers

Table 8: Vendor Affirmed Operational Environments

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

6.3 Operational Environment Policy

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

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

In operational mode, the ptrace(2) system call, the debugger (gdb(1)), 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.

7. Roles, Services and Authentication

7.1 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.
- **Crypto Officer Role**: performs module installation.

The User and Crypto Officer roles are implicitly assumed by the entity accessing services implemented by the module.

7.2 FIPS Approved Operator Services and Descriptions

The below table provides a full description of FIPS Approved services provided by the module and lists the roles allowed to invoke each service. In the table below, the "U" represents a User Role, and "CO" denotes a Crypto Officer role.

U	CO	Service Name	Service Description	Keys and CSP(s)	Access Type(s)
X		Symmetric Encryption/Decryption	Encrypts or decrypts a block of data using 3-Key Triple-DES or AES	AES or 3-Key Triple-DES Key	R, X
X		Symmetric Key Generation	Generate AES, Triple-DES, HMAC keys	AES, Triple-DES, HMAC	R, W, X
X		Key Wrapping	Key wrapping with AES-GCM KTS, AES-CBC + HMAC KTS, Triple-DES-CBC + HMAC KTS	AES, Triple-DES, HMAC	R, X
X		Asymmetric Key Generation	Generate RSA, DSA, ECDSA keys	RSA, DSA, ECDSA	R, W, X
X		Digital Signature Generation and Verification	Sign and verify operations with X509 certificates	RSA, DSA, and ECDSA keys	R, X
X		DSA Domain Parameter Generation/Verification	Generate and verify DSA domain parameters	DSA private and public keys	R, W, X
X		Public Key Verification	Verifies a public key is valid	ECDSA public key	R, X
X		Diffie-Hellman Parameters Generation using Safe Primes, Import and Export	Generate, import, and export Diffie-Hellman parameters	Diffie-Hellman domain parameters	R, W, X
X		Import and Export Public Key	Import and export public key components	RSA, DSA, ECDSA public key	R, X
X		Import and Export Private Key	Import and export private key components	RSA, DSA, ECDSA private key	R, X
X		Keyed Hash (HMAC)	HMAC-SHA services	HMAC keys > 112 bits	R, X
X		Hash	SHA hashing services	None	N/A
X		Key Derivation	TLS KDF	TLS Derived Key	R, W, X

U	CO	Service Name	Service Description	Keys and CSP(s)	Access Type(s)
			PBKDF	PBKDF password and PBKDF derived key	
			HKDF	Shared secret, HKDF derived key	
X		Shared Secret Computation	Shared secret computation	Shared secret	R, W, X
X		Random Number Generation	Generate random numbers using SP 800-90A DRBG	Entropy input string and seed, DRBG Internal V and Key	R, W, X
X		TLS/DTLS Network Protocol	Provide data encryption and authentication over TLS network protocol	TLS Derived Key	X
X		TLS/DTLS Key Agreement	Negotiate a TLS key agreement secure channel	TLS Derived Key, RSA, DSA or ECDSA keys, shared secret (pre-master secret), master secret, Diffie-Hellman and EC Diffie-Hellman Private keys	R, W, X
X		TLS Handshaking Using X509 Certificates	TLS Handshaking using X509 Certificates Authentication method with: • Diffie-Hellman KAS • EC Diffie-Hellman KAS • RSA-based PKCSv1.5 Key Wrapping	TLS Derived Key, RSA, DSA or ECDSA keys, shared secret (pre-master secret), master secret, Diffie-Hellman and EC Diffie-Hellman Private keys	R, W, X
X		TLS Handshaking Using Anonymous Authentication method	TLS Handshaking using Anonymous Authentication method with: • Diffie-Hellman KAS • EC Diffie-Hellman KAS	TLS Derived Key, RSA, DSA or ECDSA keys, shared secret (pre-master secret), master secret, Diffie-Hellman and EC Diffie-Hellman Private keys	R, W, X
X		TLS Handshaking using Pre-Shared Key (PSK) Authentication	TLS Handshaking using Pre-Shared Key (PSK) Authentication method with: • Diffie-Hellman KAS • EC Diffie-Hellman KAS • RSA-based PKCSv1.5 Key Wrapping	TLS Derived Key, RSA, DSA or ECDSA keys, shared secret (pre-master secret), master secret, Diffie-Hellman and EC Diffie-Hellman Private keys	R, W, X
X		TLS X.509 Certificate Handling, including digital signature, key/certificate import and export	TLS X.509 Certificate Handling, including digital signature, key/certificate import and export, and support the following format: • PKCS#7 • PKCS#12	RSA, DSA or ECDSA keys	R, W

U	CO	Service Name	Service Description	Keys and CSP(s)	Access Type(s)
			• Binary (DER) encoding • ASCII (PEM) encoding		
X		Show Status	Show status of the module state	None	N/A
X		Self-Test	Initiate power-on self-tests	None	N/A
X		Zeroization	Zeroize all critical security parameters	All keys and CSP's	Z
	X	Module Installation	Installation of the module	None	N/A

R – Read, W – Write, X – Execute, Z - Zeroize

Table 9: FIPS Approved Services

7.3 Non-FIPS Approved Services and Descriptions

The following table lists the non-Approved services available in non-FIPS mode. Security services listed in the table below, make use of non-approved cryptographic algorithms. Use of any of these services in Table 10 will put the module in the non-Approved mode implicitly. The algorithms associated with these services are specified in section 3.6.

U	CO	Service Name	Service Description	Keys	Access Type(s)
X		Asymmetric Encryption/Decryption	Encrypts or decrypts using non-Approved RSA key size	RSA key wrapping with keys less 2048 bits	R, X
X		AEAD Encryption/Decryption using Chacha20 and Poly1305	Authenticated encryption or decryption using ChaCha20 and Poly1305	Chacha20 key	R, X
X		Symmetric Encryption/Decryption	Encrypts or decrypts using non-Approved algorithms	AES-SIV, Camellia, ChaCha20, DES, RC2, RC4, Salsa-20 keys	R, W, X
X		Digital Signature Generation	Sign operations with non-Approved keys or non-approved message digest or SHA-1	RSA key lengths of < 2048 and key lengths of > 4096, DSA key lengths of < 2048 and key lengths of > 3072, ECDSA curve P-192 and P-224	R, X
X		Digital Signature Verification	Verify operations with non-Approved keys or non-approved message digest	RSA keys < 2048 and RSA keys > 4096, DSA keys < 2048 bits and DSA keys > 3072, ECDSA curve P-192 and P-224	R, X
X		Asymmetric signature generation/verification	Asymmetric signature using Ed25519 curve, EdDSA	EC private key	R, W, X
X		TLS/DTLS Key Agreement	Negotiate a TLS key agreement with non-	Diffie-Hellman key lengths < 2048,	R, W, X

U	CO	Service Name	Service Description	Keys	Access Type(s)
			Approved keys	Diffie-Hellman with keys generated with domain parameters other than safe primes	
X		Asymmetric Key Generation	FIPS 186-4 key generation of non-Approved RSA, DSA and ECDSA keys	RSA keys < 2048 and keys > 4092, DSA keys < 2048 and keys > 3072, ECDSA curve P-192 and P-224	R, W, X
			FIPS 186-2 key generation	RSA, DSA keys	R, W, X
X		Hash	Hashing using non-Approved hash functions that include MD2, MD4, MD5, GOST, RIPEMD-160, Streebog-256 and Streebog-512	None	N/A
X		UMAC	MAC generation using UMAC	MAC Key	R, X
X		Support to use DANE Certificate	DNS-based Authentication of Named Entities (DANE) is a protocol to allow X.509 certificates, commonly used for TLS.	RSA, DSA and ECDSA public/private keys	R, X
X		Support to use OpenPGP Certificate	Use of OpenPGP certificates for a TLS session	RSA, DSA and ECDSA public/private keys	R, X
X		Support to use PKCS#11 Certificate	Use of PKCS#11 certificates in GnuTLS	RSA, DSA and ECDSA public/private keys	R, X
X		Support to use the Secure RTP (SRTP) defined in RFC5764	a SRTP extension for DTLS	AES and HMAC keys	R X
X		Support to use Trusted Platform Module (TPM)	TPM operations supported and used by GnuTLS limited to TPM 1.2.	RSA, DSA and ECDSA public/private keys	R, X

Table 10: Non-FIPS Approved Services

Note: The module does not share CSPs between FIPS mode of operation and a non-FIPS mode of operation. All cryptographic keys used in the FIPS mode of operation must be generated in the FIPS mode or imported while running in the FIPS mode. The DRBG shall not be used for key generation for non-Approved services in non-FIPS mode.

7.4 Operator Authentication

The module does not support operator authentication mechanisms.

8. Key and CSP Management

The following keys, cryptographic key components and other critical security parameters are contained in the module.

CSP Name	Generation	Entry/Output	Storage	Zeroization
AES Keys (128, 192, 256 bits)	Keys can be generated by SP 800-90A DRBG as mentioned in section 8.2.	The key is passed into the module via API input parameters. No output mechanism provided.	RAM in plaintext	gnutls_cipher_deinit()
AES CCM/CMAC/GCM/ GMAC/XTS key (128, 256 bits)				
Triple-DES Keys (192 bits)			RAM in plaintext	gnutls_cipher_deinit()
HMAC Key (≥ 112 bits)			RAM in plaintext	gnutls_hmac_deinit()
DSA Private Key (2048 and 3072 bits)	Keys are generated using FIPS 186-4 and the random value used in the key generation is generated using SP800-90A DRBG	The key is passed into the module to and from calling application via API parameters.	RAM in plaintext	gnutls_privkey_deinit() or gnutls_x509_privkey_deinit()
ECDSA Private Key (P-256, P-384, P-521)		The key is passed into the module to and from calling application via API parameters.	RAM in plaintext	gnutls_privkey_deinit() Or gnutls_x509_ privkey_deinit()
RSA Private Key (2048, 3072, 4096 bits)		The key is passed into the module to and from calling application via API parameters.	RAM in plaintext	gnutls_rsa_params_deinit (, gnutls_privkey_deinit() or gnutls_x509 _ privkey_deinit()
Entropy Input String for DRBG Seed	Obtained from the CPU Jitter source	The module does not import or export the key or CSP.	RAM in plaintext	gnutls_global_deinit()
DRBG seed, internal state (V and Key)	Generated internally in the DRBG		RAM in plaintext	gnutls_global_deinit()
TLS Pre-Master Secret	Generated using in the Diffie-Hellman or EC Diffie-Hellman key agreement or RSA key transport during handshake.	Entry: if received by the module as TLS server, wrapped with server's public RSA key, otherwise no entry. Output: encrypted with RSA only when TLS with RSA key wrapping is used if sent by the module as TLS	RAM in plaintext	gnutls_deinit()

CSP Name	Generation	Entry/Output	Storage	Zeroization
		client.		
Shared Secret	Generated by the module in the Diffie-Hellman or EC Diffie-Hellman shared secret computation.	The key is passed into the module to and from calling application via API parameters.	RAM in plaintext	gnutls_deinit()
Master secret	Derived from pre-master secret.	Generated inside the module. No output mechanism provided.	RAM in plaintext	gnutls_deinit()
TLS Derived Key	Generation during TLS KDF	Generated inside the module. No output mechanism provided.	RAM in plaintext	Internal state is zeroized automatically when function returns.
Diffie-Hellman keys 2048, 3072, 4096, 6144 and 8192 bits	Keys are generated using SP 800-56Arev3 and the random value used in the key generation is generated using SP 800-90A DRBG.	The key is passed into the module to and from calling application via API parameters.	RAM in plaintext	gnutls_deinit() or gnutls_dh_params_deinit()
EC Diffie-Hellman private keys P-256, P-384 and P-521 curves	Keys are generated using FIPS 186-4 and the random value used in the key generation is generated using SP 800-90A DRBG.	The key is passed into the module to and from calling application via API parameters.	RAM in plaintext	gnutls_deinit() or gnutls_ecdh_params_deinit()
PBKDF derived key	Generated during PBKDF	The resulting key is output through output parameters.	RAM in plaintext	Internal PBKDF state is zeroized automatically when function returns.
PBKDF password	N/A	The password is passed into the module via API input parameters.	RAM in plaintext	Internal PBKDF state is zeroized automatically when function returns.
HKDF derived key	Generated during HKDF	The resulting key is output through output parameters.	RAM in plaintext	Internal HKDF state is zeroized automatically when function returns.

Table 11: Keys/CSPs Table

8.1 Random Number Generation

The module provides an SP 800-90A-compliant Deterministic Random Bit Generator (DRBG) for creation of key components of asymmetric keys, and random number generation.

The seeding (and automatic reseeding) of the DRBG is done with getrandom().

The module employs the Deterministic Random Bit Generator (DRBG) based on [SP 800-90A] for the random number generation. The DRBG supports the CTR_DRBG mechanisms. The module performs the DRBG health tests as defined in section 11.3 of [SP 800-90A]. The module uses CPU jitter as a noise source provided by the operational environment which is within the module's physical boundary but outside of the module's logical boundary. The source is compliant with [SP 800-90B] and marked as ENT on the certificate. The entropy provided from the entropy source provides 230 bits of entropy and the module requires up to 256 bits of entropy. The caveat, "The module generates cryptographic keys whose strengths are modified by available entropy" applies.

8.2 Key Generation

The Key Generation methods implemented in the module for Approved services in FIPS mode is compliant with [SP 800-133].

For generating RSA, DSA and ECDSA keys the module implements asymmetric key generation services compliant with [FIPS 186-4] and [SP 800-90A]. A seed (i.e. the random value) used in asymmetric key generation is directly obtained from the [SP 800-90A] DRBG.

The public and private key pairs used in the Diffie-Hellman and EC Diffie-Hellman KAS are generated internally by the module using key generation compliant with [SP 800-56Arev3].

The module supports the generation of symmetric keys. gnutls_rnd() can be used to generate symmetric keys. Each will call the DRBG compliant to [SP 800-90A] to generate the key materials for symmetric keys or HMAC keys. Therefore, CKG (vendor affirmed) is mentioned on the draft certificate.

The module generates cryptographic keys whose strengths are modified by available entropy.

8.3 Key Establishment / Key Derivation

The module provides Diffie-Hellman and EC Diffie-Hellman shared secret computation compliant with SP 800-56Arev3, in accordance with scenario X1 (1) of IG D.8 with Diffie-Hellman with at least 2048 bits key size and EC Diffie-Hellman with P-256, P-384 or P-521 curve in FIPS mode. The Diffie-Hellman with less than 2048 bits key size is only available in non-FIPS mode.

The module provides Diffie-Hellman and EC Diffie-Hellman key agreement schemes compliant with SP 800-56rev3 and used as part of the TLS protocol key exchange in accordance with scenario X1 (2) of IG D.8; that is, the shared secret computation (KAS-FFC-SSC and KAS-ECC-SSC) followed by the derivation of the keying material using SP 800-135 KDF.

For Diffie-Hellman, the module supports the use of safe primes from RFC 7919 for domain parameters and key generation, which are used in the TLS key agreement implemented by the module.

TLS (RFC 7919)

- ffdhe2048 (ID = 256)
- ffdhe3072 (ID = 257)
- ffdhe4096 (ID = 258)
- ffdhe6144 (ID = 259)
- ffdhe8192 (ID = 260)

The module also supports RSA key wrapping using encryption and decryption primitives with the modulus size of at least 2048 bits in FIPS mode. The modulus size of 1024 bits is only available in non-FIPS mode. According to Table 2: Comparable strengths in NIST SP 800-57 Part1 Rev5 (dated on May, 2020), the key sizes of RSA, Diffie-Hellman and EC Diffie Hellman provides the following security strength for the corresponding key establishment method shown below:

- RSA key wrapping provides between 112 and 256 bits of encryption strength;
- Diffie-Hellman key agreement provides between 112 and 200 bits of encryption strength;
- Diffie-Hellman shared secret computation provides between 112 and 200 bits of encryption strength;
- EC Diffie-Hellman key agreement provides between 128 and 256 bits of encryption strength;
- EC Diffie-Hellman shared secret computation provides between 128 and 256 bits of encryption strength.

The module provides approved key transport methods compliant to SP 800-38F according to IG D.9. The key transport method is provided by:

- AES-GCM and AES-CCM
- AES-CBC with HMAC used within the TLS protocol.
- Triple-DES-CBC with HMAC used within the TLS protocol.
- Therefore, the following caveats apply:
  - KTS (AES Certs. #A1704, #A1705, #A1710 and #A2560; key establishment methodology provides 128 or 256 bits of encryption strength)
  - KTS (AES Certs. #A1704, #A1705, #A1710 and #A2560 and HMAC Certs. #A1705, #A1710 and #A2560; key establishment methodology provides 128 or 256 bits of encryption strength)
  - KTS (Triple-DES Cert. #A1710 and HMAC Certs. #A1705, #A1710 and #A2560; key establishment methodology provides 112 bits of encryption strength)

Note: As the module supports the RSA key pair with 16384 bits or more modulus size, the encryption strength 256 bits is claimed for RSA key wrapping.

The module supports the following key derivation methods according to [SP 800-135]:

- KDF for the TLS protocol, used as pseudo-random functions (PRF) for TLSv1.0/1.1 and TLSv1.2.

The module supports the following derivation methods according to [SP 800-56C1]:

- HKDF for the protocol TLSv1.3.

The module also supports password-based key derivation (PBKDF). The implementation is compliant with option 1a of [SP 800-132]. Keys derived from passwords or passphrases using this method can only be used in storage applications.

8.4 Key Entry and Output

The module does not support manual key entry or intermediate key generation key output. For symmetric algorithms or for HMAC, the keys are provided to the module via API input parameters for the cryptographic operations. For asymmetric algorithms, the keys are also provided to the module via API input parameters. The module also provides the services to import and export public and private keys to and from calling application only.

8.5 Key / CSP Storage

The module does not support persistent key storage. The key and CSPs are stored as plaintext in the RAM. The keys are provided to the module via API input parameters, and are destroyed by the module using appropriate API function calls before they are released in the memory.
The HMAC key used for integrity test is stored in the .hmac file and relies on the operating system for protection.

8.6 Key / CSP Zeroization

The memory occupied by keys is allocated by regular libc malloc/calloc() calls. The application that uses the module is responsible for calling the appropriate destruction functions from the GnuTLS API to zeroize the keys or keying material. The destruction functions then overwrite the memory occupied by keys with pre-defined values and deallocates the memory with the free() call. In case of abnormal termination, or swap in/out of a physical memory page of a process, the keys in physical memory are overwritten by the Linux kernel before the physical memory is allocated to another process.

9. Self-Tests

FIPS 140-2 requires that the module perform power-up tests to ensure the integrity of the module and the correctness of the cryptographic functionality at start up. In addition, some functions require continuous testing of the cryptographic functionality, such as the asymmetric key generation. If any self-test fails, the module returns an error code and enters the error state. No data output or cryptographic operations are allowed in error state. See section 10.3 for descriptions of possible self-test errors and recovery procedures.

9.1 Power-Up Self-Tests

The module performs power-up self-tests automatically when the module is loaded into memory; power-up tests ensure that the module is not corrupted and that the cryptographic algorithms work as expected. Input, output, and cryptographic functions cannot be performed while the module is in a self-test state because the module is single-threaded and will not return to the calling application until the power-up self-tests are completed. If any power-up self-test fails, the module returns the error code listed in section 10.3 and displays “Error in GnuTLS initialization” and then enters error state. The subsequent calls to the module will also fail - thus no further cryptographic operations are possible. If the power-up self-tests complete successfully, the module will return 0 and accepts cryptographic operation services request.

9.1.1 Integrity Tests

The integrity of the module is verified by comparing an HMAC-SHA-256 value calculated at run time with the HMAC value stored in the .hmac file that was computed at build time for each component of the module. If the HMAC values do not match, the test fails and the module enters the error state.

9.1.2 Cryptographic Algorithms Tests

The module performs self-tests on all FIPS-Approved cryptographic algorithms supported in the approved mode of operation, using the known answer tests (KAT) and pair-wise consistency test (PCT), shown in the following table:

Algorithm	Test
AES	KAT AES-CBC/GCM/CCM/CMAC encryption and decryption are tested separately.
Triple-DES	KAT Triple-DES-CBC encryption and decryption are tested separately.
HMAC	(SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) KAT
SHA	Tested as part of HMAC KAT according to IG 9.1 SHA3-224, SHA3-256, SHA3-384, SHA3-512 KATs
DSA	sign and verify KAT with 2048 bit keys and SHA-256.
RSA	sign and verify KAT with 2048 bit keys and SHA-256.
ECDSA	sign and verify KAT with curves P-256 and SHA-256
Diffie-Hellman	Primitive "Z" Computation KAT with 3072-bit key
EC Diffie-Hellman	Primitive "Z" Computation KAT with P-256 curve
DRBG	CTR_DRBG KAT with AES-256
DRBG	DRBG health tests as specified in section 11.3 of NIST SP 800-90Ar1
Module Integrity	HMAC-SHA-256
TLS KDF	KAT with SHA-256
PBKDF	KAT with SHA-256

HKDF	KAT with SHA-256

Table 12: Power-On Self-Tests

For the KAT, the module calculates the result and compares it with the known value. If the answer does not match the known answer, the KAT is failed and the module returns the error code and enters the error state. As described in section 3.3, only one AES or SHA implementation from libnettle library written in C language or using the support from AES-NI or SSSE3 instructions is available at run-time. The KATs cover different implementations depending on the implementations availability in the operating environment.

9.2 On Demand Self-Tests

The on-demand self-tests is invoked by powering-off and reloading the module which cause the module to run the power-up tests again. During the execution of the on-demand self-tests, services are not available and no data output or input is possible.

9.3 Conditional Self-Tests

The module performs conditional tests on the cryptographic algorithms, using the pair-wise consistency test (PCT) shown in the following table:

Algorithm	Test
DSA	Pairwise Consistency Test: signature generation and verification
ECDSA	Pairwise Consistency Test: signature generation and verification
RSA	Pairwise Consistency Test: signature generation and verification, encryption and decryption

Table 13: Conditional Self-Tests

10. Crypto-Officer and User Guidance

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

10.1 Crypto-Officer Guidance

The version of the RPM containing the validated module is stated in section 3.1 above. The RPM package of the Module can be installed by standard tools recommended for the installation of Oracle packages on an Oracle Linux system (for example, yum, RPM, and the RHN remote management tool). 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 Oracle Linux Yum Server indicates an integrity error. The RPM files listed in section 3 are signed by Oracle and during installation; Yum performs signature verification which ensures as secure delivery of the cryptographic module. If the RPM packages are downloaded manually, then the CO should run 'rpm –K <rpm-file-name>' command after importing the builder's GPG key to verify the package signature. In addition, the CO can also verify the hash of the RPM package to confirm a proper download.

Recommended method

The system-wide cryptographic policies package (crypto-policies) contains a tool that completes the installation of cryptographic modules and enables self-checks in accordance with the requirements of Federal Information Processing Standard (FIPS) Publication 140-2. We call this step "FIPS enablement". The tool named fips-mode-setup installs and enables or disables all the validated FIPS modules and it is the recommended method to install and configure an Oracle Linux 8 system.

1. Ensure that the OL8 x86_64 or aarch64 system is configured with "Latest" and "Security Validation (Update 4)" yum repositories enabled, for example:
   # yum-config-manager --enable ol8_latest ol8_u4_security_validation
   Note: If system is configured with the Unbreakable Linux Network (ULN) depending on the architecture make sure enabled channels [ol8_x86_64_latest, ol8_x86_64_u4_security_validation] for x86_64 or [ol8_aarch64_latest, ol8_aarch64_u4_security_validation] for aarch64.

2. Install GnuTLS RPM file from the yum/ULN:
   # yum install gnutls-3.6.14-8.0.1.el8.x86_64.rpm or gnutls-3.6.14-8.0.1.el8.aarch64.rpm

FIPS enablement using fips-mode-setup tool

1. Switch the system to FIPS enablement in Oracle Linux 8:
   # fips-mode-setup --enable
   Setting system policy to FIPS
   FIPS mode will be enabled.
   Reboot the system for the setting to take effect.
2. Restart your system:
   # reboot
3. After the restart, you can check the current state:
   # fips-mode-setup --check
   FIPS mode is enabled.

Note: As a side effect of the enablement procedure the fips-mode-enable tool also changes the system wide cryptographic policy level to a level named "FIPS", this level helps applications by changing configuration defaults to approved algorithms.

FIPS enablement via environment variable

1. Open:
   /etc/bashrc
2. Set:
   export GNUTLS_FORCE_FIPS_MODE=1
3. Save and close.
4. Run:
   # source /etc/bashrc

10.1.1 AES Hardware Acceleration Support and Manual Method

According to the OpenSSL FIPS 140-2 Security Policy, the OpenSSL module supports the AES-NI Intel processor instruction and ARM AES optimizations set as an approved cipher. Both architecture optimizations are used by the Module.

In case you configured a full disk encryption using AES, you may use the aforementioned optimizations for a higher performance compared to the software-only implementation.

Verify that your processor offers AES hardware acceleration by calling the following command:
cat /proc/cpuinfo | grep aes

If the command returns a list of properties, including the “aes” string, your CPU provides the AES hardware acceleration. If the command returns nothing, AES hardware acceleration is not supported.

The recommended method automatically performs all the necessary steps. The following steps can be done manually but are not recommended and are not required if the systems has been installed with the fips-mode-setup tool:

- Create a file named /etc/system-fips, the contents of this file are never checked
- Ensure to invoke the command ‘fips-finish-install --complete’ on the installed system
- Ensure that the kernel boot line is configured with the fips=1 parameter set
- Reboot the system

NOTE: If /boot or /boot/efi resides on a separate partition, the kernel parameter boot=<boot partition> must be supplied. The partition can be identified with the command "df | grep boot". For example:

```
$ df  | grep boot
```

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

The partition of the /boot file system is located on /dev/sda1 in this example.

Therefore the parameter boot=/dev/sda1 needs to be appended to the kernel command line in addition to the parameter fips=1.

10.2 User Guidance

The applications must be linked dynamically to run the module. Only the services listed in Table 9 are allowed to be used in FIPS mode.

The libraries of GMP and Nettle provides the support of cryptographic operations to the GnuTLS library. The operator shall use the API provided by the GnuTLS library for the services. Invoking the APIs provided by the supporting libraries are forbidden.

10.2.1 TLS and Diffie-Hellman

The TLS protocol implementation provides both, the server and the client sides. As required by SP 800-131A, For Diffie-Hellman only the safe prime groups listed in RFC7919 are approved to be used in FIPS mode. The TLS protocol cannot enforce the support of FIPS Approved Diffie-Hellman key sizes. To ensure full support for all TLS protocol versions, the TLS client implementation of the cryptographic module must accept Diffie-Hellman key sizes smaller than 2048 bits offered by the TLS server.

The TLS server implementation of the cryptographic Module allows the application to set the Diffie-Hellman key size. The server side must always set the DH parameters with the API call of:

SSL_CTX_set_tmp_dh(ctx, dh)

Alternatively it is possible to use SSL_CTX_set_dh_auto(ctx, 1); function call that makes GnuTLS use built-in 2048 bit parameters when the server RSA certificate is at least 2048 bits and 3072 bit DH parameters with RSA certificate of 3072 bits.

To comply with the FIPS 140-2 standard the requirement to not allow Diffie-Hellman key sizes smaller than 2048 bits must be met, to do this the Crypto Officer must ensure that:

- in case the Module is used as TLS server, the Diffie-Hellman parameters (dh argument) of the aforementioned API call must be 2048 bits or larger;
- in case the Module is used as TLS client, the TLS server must be configured to only offer Diffie-Hellman keys of 2048 bits or larger.

10.2.2 AES GCM IV Guidance

In case the module's power is lost and then restored, the key used for the AES GCM encryption or decryption shall be re-distributed. The AES GCM IV generation is in compliance with the [RFC 5288] and shall only be used for the TLS protocol version 1.2 to be compliant with [FIPS140-2_IG] IG A.5; thus, the module is compliant with [SP 800 52]. If the nonce_explicit part of the IV exhausts, GnuTLS will mark the TLS session as invalid and the key will need to be renegotiated.

10.2.3 AES-XTS Guidance

The length of a single data unit encrypted or decrypted with the AES-XTS shall not exceed 2^{20} AES blocks that is 16MB of data per AES-XTS instance. An XTS instance is defined in section 4 of SP 800-38E. The AES-XTS mode shall only be used for the cryptographic protection of data on storage devices. The AES-XTS shall not be used for other purposes, such as the encryption of data in transit. The module implements the check to ensure that the two AES keys used in XTS-AES algorithm are not identical.

10.2.4 RSA and DSA Keys

As per SP 800-131A, RSA and DSA must be used at least 2048 bit keys in FIPS mode. To comply with the requirements of [FIPS 140-2], the operator must therefore only use keys with at least 2048 bits in FIPS mode.

10.2.5 Symmetric Key Generation

The API function gnutls_key_generate() shall not be used in FIPS mode of operation. The caller shall call gnutls_rnd() which calls the DRBG compliant to [SP 800-90A] to generate the key materials for symmetric keys or HMAC keys.

10.2.6 Triple-DES

According to IG A.13, the same Triple-DES key shall not be used to encrypt more than 2^{16} 64- bit blocks of data.

10.2.7 Key derivation using SP800-132 PBKDF

The module provides password-based key derivation (PBKDF), 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], the following requirements shall be met.

- Derived keys shall only be used in storage applications. The Master Key (MK) shall not be used for other purposes. The length of the MK or DPK shall be of 112 bits or more.
- A portion of the salt, with a length of at least 128 bits, shall be generated randomly using the SP 800-90A DRBG.
- The iteration count shall be selected as large as possible, as long as time required to generate the key using entered password is acceptable for the users. The minimum value shall be 1000.
- Passwords or passphrases, used as an input for PBKDF, shall not be used as cryptographic keys.

The length of the password or passphrase shall be of at least 20 characters, and shall consist of lower-case, upper-case and numeric characters. The probability of guessing the value is estimated to be $1/62^{20} = 10^{-36}$, which is less than 2^{-112}.

The calling application shall also observe the rest of the requirements and recommendations specified in [SP 800-132].

10.3 Handling Self-Test Errors

When the module fails any self-test, it will return an error code to indicate the error and enters error state that any further cryptographic operations is inhibited. Here is the list of error codes when the module fails any self-test or in error state:

Error Events	Error Codes	Error Messages
When the KAT or Integrity fails at the power-up	GNUTLS_E_SELF_TEST_ERROR (-400)	“Error while performing self checks.”
When the KAT of DRBG fails at the power-up	GNUTLS_E_RANDOM_FAILED (-206)	“Error while performing self checks.”
When the new generated RSA, DSA or ECDSA key pair fails the PCT	GNUTLS_E_PK_GENERATION_ERROR (-403)	“Error in public key generation.”
When the module is in error state and caller requests cryptographic operations	GNUTLS_E_LIB_IN_ERROR_STATE (-402)	"An error has been detected in the library and cannot continue operations."

Table 14: Error Events and Error Messages

Self-test errors transition the module into an error state that keeps the module operational but prevents any cryptographic related operations. The module must be restarted and perform power-up self-test to recover from these errors. If failures persist, the module must be re-installed.

A completed list of the error codes can be found in Appendix C “Error Codes and Descriptions” in the gnutls.pdf provided with the module's code.

11. Mitigation of Other Attacks

RSA is vulnerable to timing attacks. In a setup where attackers can measure the time of RSA decryption or signature operations, blinding is always used to protect the RSA operation from that attack.

The internal API function of _rsa_blind() and _rsa_unblind() are called by the module for RSA signature generation and RSA decryption operations. The module generates a random blinding factor and include this random value in the RSA operations to prevent RSA timing attacks.

Acronyms, Terms and Abbreviations

Term	Definition
AES	Advanced Encryption Standard
AES-NI	Advanced Encryption Standard New Instructions
API	Application Program Interface
CAVP	Cryptographic Algorithm Validation Program
CBC	Cypher Block Chaining
CFB	Cipher Feedback
CKG	Cryptographic Key Generation
CMVP	Cryptographic Module Validation Program
CSE	Communications Security Establishment
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DH	Diffie-Hellman
DHE	Diffie-Hellman Ephemeral
DRBG	Deterministic Random Bit Generator
DTLS	Datagram Transport Layer Security
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
FFC	Finite Field Cryptography
GCM	Galois Counter Mode
GPC	General Purpose Computer
HMAC	(Keyed) Hash Message Authentication Code
IG	Implementation Guidance
KAS	Key Agreement Schema
KAT	Known Answer Test
KDF	Key Derivation Function
NIST	National Institute of Standards and Technology
PAA	Processor Algorithm Acceleration
PBKDF	Password Based Key Derivation Function
PCT	Pair-Wise Consistency Test
PR	Prediction Resistance
PRF	Pseudo Random Function
PSS	Probabilistic Signature Scheme
PUB	Publication
RPM	Red Hat Package Manager
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SP	Special Publication
SSSE	Supplemental Streaming SIMD Extensions 3
TLS	Transport Layer Security

Table 15: Acronyms

References

The FIPS 140-2 standard, and information on the CMVP, can be found at http://csrc.nist.gov/groups/STM/cmvp/index.html. More information describing the module can be found on the Oracle web site at https://www.oracle.com/linux/ .

This Security Policy contains non-proprietary information. All other documentation submitted for FIPS 140-2 conformance testing and validation is "Oracle - Proprietary" and is releasable only under appropriate non-disclosure agreements.

Document	Author	Title
FIPS PUB 140-2	NIST	FIPS PUB 140-2: Security Requirements for Cryptographic Modules
FIPS IG	NIST	Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program
FIPS PUB 140-2 Annex A	NIST	FIPS 140-2 Annex A: Approved Security Functions
FIPS PUB 140-2 Annex B	NIST	FIPS 140-2 Annex B: Approved Protection Profiles
FIPS PUB 140-2 Annex C	NIST	FIPS 140-2 Annex C: Approved Random Number Generators
FIPS PUB 140-2 Annex D	NIST	FIPS 140-2 Annex D: Approved Key Establishment Techniques
DTR for FIPS PUB 140-2	NIST	Derived Test Requirements (DTR) for FIPS PUB 140-2, Security Requirements for Cryptographic Modules
FIPS PUB 197	NIST	Advanced Encryption Standard
FIPS PUB 198-1	NIST	The Keyed Hash Message Authentication Code (HMAC)
FIPS PUB 186-4	NIST	Digital Signature Standard (DSS)
FIPS PUB 180-4	NIST	Secure Hash Standard (SHS)
NIST SP 800-52	NIST	Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations
NIST SP 800-56Arev3	NIST	Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography
NIST SP 800-56C	NIST	Recommendation for Key-Derivation Methods in Key-Establishment Schemes
NIST SP 800-67	NIST	Recommendation for the Triple Data Encryption Algorithm TDEA Block Cipher
NIST SP 800-90A	NIST	Recommendation for Random Number Generation Using Deterministic Random Bit Generators
NIST SP 800-90B	NIST	Recommendation for the Entropy Sources Used for Random Bit Generation
NIST SP 800-131A	NIST	Recommendation for the Transitioning of Cryptographic Algorithms and Key Sizes
NIST SP 800-133	NIST	Recommendation for Cryptographic Key Generation
NIST SP 800-135	NIST	Recommendation for Existing Application-Specific Key Derivation Functions
PKCS#1	RSA Laboratories	PKCS#1 v2.1: RSA Cryptographic Standard
RFC4347	IETF	Datagram Transport Layer Security
RFC4357	IETF	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

Document	Author	Title
RFC5288	IETF	AES Galois Counter Mode (GCM) Cipher Suites for TLS
RFC5246	IETF	The Transport Layer Security (TLS) Protocol Version 1.2
RFC5764	IETF	Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)

Table 16: References