# Red Hat Enterprise Linux 9 gnutls

# version 3.7.6-66803fa128d6a6e5

# FIPS 140-3 Non-Proprietary Security Policy

**document version 1.1**

**Last update: 2024-08-09**

Prepared by:

atsec information security corporation

4516 Seton Center Parkway, Suite 250

Austin, TX 78759

[www.atsec.com](http://www.atsec.com)

**Table of Contents**

# 1  General

## 1.1 Overview

This document is the non-proprietary FIPS 140-3 Security Policy for version 3.7.6-66803fa128d6a6e5 of the Red Hat Enterprise Linux 9 gnutls cryptographic module. It has a one-to-one mapping to the [SP 800-140B] starting with section B.2.1 named "General" that maps to section 1 in this document and ending with section B.2.12 named "Mitigation of other attacks" that maps to section 12 in this document. 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.

## 1.2 Security Levels

| 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 |
| **Overall** | | **1** |

*Table 1 - Security Levels*

# 2 Cryptographic Module Specification

## 2.1 Description

The Red Hat Enterprise Linux 9 gnutls cryptographic module (hereafter referred to as "the module") is a software library. The module is an open-source, general-purpose set of libraries designed to support cross-platform development of security-enabled client and server applications. The module is a multiple-chip standalone cryptographic module.

## 2.2 Version Information

The module version is 3.7.6-66803fa128d6a6e5 of the Red Hat Enterprise Linux 9 gnutls cryptographic module.

## 2.3 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® Xeon® 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 with PowerVM FW1040.00 with VIOS 3.1.3.00 | IBM 9080-HEX | IBM POWER10 | ISA |

*Table 2 - Tested Operational Environments*

In addition to the configurations tested by the atsec CST laboratory, vendor-affirmed testing was performed on the following platforms for the module by F5, Inc.

| # | Operating System | Hardware Platform |
|---|---|---|
| 1 | Red Hat Enterprise Linux 9 | Intel® Xeon® E5 |

*Table 3 - Vendor Affirmed Operation Environments*

Note: The 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.

| Component | Description |
|---|---|
| /usr/lib64/libgnutls.so.30 Note: libgmp is statically linked to libgnutls | Provides the API for the calling applications to request cryptographic services, and implements the TLS protocol, DRBG, RSA Key Generation, Diffie-Hellman and EC Diffie-Hellman. |
| /usr/lib64/libnettle.so.8 | Provides the cryptographic algorithm implementations, including AES, SHA, HMAC, RSA Digital Signature, DSA and ECDSA. |
| /usr/lib64/libhogweed.so.6 | Provides primitives used by libgnutls and libnettle to support the asymmetric cryptographic operations. |
| /usr/lib64/.libgnutls.so.30.hmac | The .hmac file contain the HMAC-SHA2-256 values of the libraries for integrity check during the power-up. |

*Table 4 – Cryptographic Module Components*

## 2.4 Modes of Operations

When the module starts up successfully, after passing all the pre-operational and conditional cryptographic algorithms self-tests (CASTs), the module is operating in the approved mode of operation by default and can only be transitioned into the non-Approved mode by calling one of the non-Approved services listed in Table 10. Please see section 4 for the details on service indicator provided by the module that identifies when an approved service is called.

## 2.5 Approved Algorithms

The table below lists all security functions of the module, including specific key size(s) employed for approved or vendor-affirmed security functions, and implemented modes of operation.

| CAVP Cert | Algorithm and Standard | Mode/Method | Description/Key Size(s)/Key Strength(s) | Use/Function |
|---|---|---|---|---|
| Certs. #A3472, #A3473, #A3478, #A3550, #A3551 | AES FIPS197, SP800-38A | CBC | 128, 192, 256-bit keys with 128-256 bits key strength | Symmetric encryption; Symmetric decryption |
| Cert. #A3478 | AES FIPS197, SP800-38A | ECB | 128, 192, 256-bit keys with 128-256 bits key strength | Symmetric encryption; Symmetric decryption |
| Certs. #A3472, #A3550 | AES SP800-38C | CCM | 128, 256-bit keys with 128 or 256 bits key strength | Symmetric encryption; Symmetric decryption; Authenticated encryption; Authenticated decryption |
| Certs. #A3475, #A3476, #A3481 | AES FIPS197, SP800-38A | CFB8 | 128, 192, 256-bit keys with 128 or 256 bits key strength | Symmetric encryption; Symmetric decryption |
| Certs. #A3472, #A3473, #A3478, #A3550 | AES SP800-38B | CMAC | 128, 256-bit keys with 128 or 256 bits key strength | Message authentication code (MAC) Message authentication code verification |
| Certs. #A3472, #A3473, #A3478, #A3550, #A3551 | AES SP800-38D | GCM | 128, 256-bit keys with 128 or 256 bits key strength | Symmetric encryption and decryption in the context of the Transport Layer Security (TLS) network protocol |
| Cert. #A3478 | AES SP800-38D | GMAC | 128, 256-bit keys with 128 or 256 bits key strength | Message authentication code (MAC) Message authentication code verification |
| Cert. #A3479 | AES SP800-38E | XTS | 256, 512-bit keys with 128 or 256 bits key strength | Symmetric encryption (for data storage); Symmetric decryption (for data storage) |

| CAVP Cert | Algorithm and Standard | Mode/Method | Description/Key Size(s)/Key Strength(s) | Use/Function |
|---|---|---|---|---|
| Vendor Affirmed | CKG SP800-133rev2 | Key pair generation (FIPS-186-4, SP800-56Arev3, SP800-90Arev1) | RSA: 2048, 3072, 4096-bit keys with 112-149 bits key strength ECDSA/ECDH: P-256, P-384, P-521 elliptic curves with 128-256 bits key strength Safe Primes: 2048, 3072, 4096, 6144, 8192-bit keys with 112-200 bits of key strength | Key pair generation |
| Cert. #A3478 | DRBG SP800-90Arev1 | CTR_DRBG: AES-256 without DF, without PR | 256-bit keys with 256 bits key strength | Random number generation |
| Cert. #A3478 | ECDSA FIPS186-4 | ECDSA KeyGen (B.4.2 Testing Candidates) | P-256, P-384, P-521 elliptic curves with 128-256 bits key strength | Key pair generation |
| | | ECDSA KeyVer | P-256, P-384, P-521 elliptic curves with 128-256 bits key strength | Public key verification |
| | | SHA-224, SHA-256, SHA-384, SHA-512 | P-256, P-384, P-521 elliptic curves with 128-256 bits key strength | Digital signature generation |
| | | SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 | P-256, P-384, P-521 elliptic curves with 128-256 bits key strength | Digital signature verification |
| Certs. #A3473, #A3478, #A3552 | HMAC FIPS198-1 | SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 | 112-524288 bit keys with 112-256 key strength | Message authentication code (MAC) Message authentication code verification |
| Cert. #A3478 | KAS-ECC-SSC SP800-56Arev3 | ECC Ephemeral Unified Scheme | P-256, P-384, P-521 elliptic curves keys with 128-256 bits key strength | EC Diffie-Hellman shared secret computation; Transport Layer Security (TLS) network protocol |
| Cert. #A3478 | KAS-FFC-SSC SP800-56Arev3 | Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 | 2048, 3072, 4096, 6144, 8192-bit keys with 112-200 bits key strength | Diffie-Hellman shared secret computation; Transport Layer Security (TLS) network protocol |
| Cert. #A3477 | KDA HKDF SP800-56Crev1 | SHA-224, SHA-256, SHA-384, SHA-512 | Derived key with 112 to 256 bits of key strength | HKDF key derivation; Transport Layer Security (TLS) network protocol |

| CAVP Cert | Algorithm and Standard | Mode/Method | Description/Key Size(s)/Key Strength(s) | Use/Function |
|---|---|---|---|---|
| Cert. #A3478 | TLS v1.2 KDF RFC7627 SP800-135rev1 (CVL) | TLS v1.2 with SHA-256, SHA-384 | Derived key with 112 to 256 bits of key strength | TLS key derivation |
| Certs. #A3472, #A3550 | AES CCM SP800-38C | KTS per IG D.G | 128, 256-bit keys with 128 or 256 bits of key strength | Key wrapping; Key unwrapping (as part of the cipher suites in the TLS protocol) |
| Certs. #A3472, #A3473, #A3478, #A3550, #A3551 | AES GCM SP800-38D | KTS per IG D.G | 128, 256-bit keys with 128 or 256 bits of key strength | |
| AES Certs. #A3472, #A3473, #A3478, #A3550, #A3551  HMAC Certs. #A3473, #A3478, #A3552 | AES CBC and HMAC SP800-38A, FIPS198-1 | KTS per IG D.G | 128, 256-bit keys with 128 or 256 bits of key strength | |
| Cert. #A3478 | PBKDF SP800-132 | HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512 | 112-256 bits 14-128 characters with password strength between $10^{14}$ and $10^{128}$ | Password-based key derivation |
| Cert. #A3478 | RSA FIPS186-4 FIPS 140-3 IG C.F | RSA KeyGen (B.3.2 Random Provable Primes) | 2048, 3072, 4096-bit keys with 112-149 bits key strength | Key pair generation |
| | | RSA SigGen PKCS#1v1.5: SHA-224, SHA-256, SHA-384, SHA-512 | 2048, 3072, 4096-bit keys with 112-149 bits key strength | Digital signature generation |
| | | RSA SigGen PSS: SHA-256, SHA-384, SHA-512 | 2048, 3072, 4096-bit keys with 112-149 bits key strength | |
| | | RSA SigVer PKCS#1v1.5: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 | 2048, 3072, 4096-bit keys with 112-149 bits key strength | Digital signature verification |
| | | RSA SigVer PSS: SHA-256, SHA-384, SHA-512 | 2048, 3072, 4096-bit keys with 112-149 bits key strength | |
| Cert. #A3478 | Safe Primes Key Generation SP800-56Arev3 | Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 | 2048, 3072, 4096, 6144, 8192-bit keys with 112-200 bits key strength | Key pair generation |

| CAVP Cert | Algorithm and Standard | Mode/Method | Description/Key Size(s)/Key Strength(s) | Use/Function |
|-----------|------------------------|-------------|------------------------------------------|--------------|
| Certs. #A3474, #A3480 | SHA-3 FIPS202 FIPS 140-3 IG C.C | SHA3-224, SHA3-256, SHA3-384, SHA3-512 | N/A | Message digest |
| Certs. #A3473, #A3478, #A3552 | SHA FIPS180-4 | SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 | N/A | Message digest |

*Table 5 - Approved Algorithms*

## 2.6 Non-Approved Algorithms

Non-Approved Algorithms Allowed in the Approved Mode of Operation:

The module does not implement any Non-Approved Algorithms Allowed in the Approved Mode of Operation.

Non-Approved Algorithms Allowed in the Approved Mode of Operation with No Security Claimed:

The module does not implement any non-Approved but Allowed algorithm in Approved mode of operation with no security claimed.

Non-Approved Algorithms Not Allowed in the Approved Mode of Operation:

The table below lists Non-Approved security functions that are not Allowed in the Approved Mode of Operation.

| Algorithm/Functions | Use/Function |
|---------------------|--------------|
| AES GCM not in the context of the TLS protocol | Symmetric encryption; Symmetric decryption |
| Blowfish | Symmetric encryption; Symmetric decryption |
| Camellia | Symmetric encryption; Symmetric decryption |
| CAST | Symmetric encryption; Symmetric decryption |
| ChaCha20 | Symmetric encryption; Symmetric decryption |
| Chacha20 and Poly1305 | Authenticated encryption; Authenticated decryption |
| DES | Symmetric encryption; Symmetric decryption |
| Diffie-Hellman with keys generated with domain parameters other than safe primes | Key agreement; Diffie-Hellman shared secret computation |
| DSA | Key pair generation; Domain parameter generation; Digital signature generation; Digital signature verification |
| ECDSA with curves not listed in Table 5 | Key pair generation; Public key verification; Digital signature generation; Digital signature verification |
| EC Diffie-Hellman with curves not listed in Table 5 | Key agreement; EC Diffie-Hellman shared secret computation |
| GOST | Symmetric encryption; Symmetric decryption; Message digest |
| HMAC with keys smaller than 112-bit | Message authentication code (MAC) |
| HMAC with GOST | Message authentication code (MAC) |
| MD2, MD4, MD5 | Message digest; Message authentication code (MAC) |

| Algorithm/Functions | Use/Function |
|---|---|
| PBKDF with non-approved message digest algorithms | Password-based key derivation |
| RC2, RC4 | Symmetric encryption; Symmetric decryption |
| RMD160 | Message digest; Message authentication code (MAC) |
| RSA with keys smaller than 2048 bits or greater than 4096 bits. | Key pair generation; Digital signature generation |
| RSA with keys smaller than 1024 bits or greater than 4096 bits. | Digital signature verification |
| RSA encryption and decryption with any key sizes. | Key encapsulation; Key un-encapsulation |
| Salsa20 | Symmetric encryption; Symmetric decryption |
| SM3 | Message digest |
| Serpent | Symmetric encryption; Symmetric decryption |
| SHA-1 | Digital signature generation |
| STREEBOG | Message digest; Message authentication code (MAC) |
| Triple-DES | Symmetric encryption; Symmetric decryption |
| Twofish | Symmetric encryption; Symmetric decryption |
| UMAC | Message authentication code (MAC) |
| Yarrow | Random number generation |

*Table 6 - Non-Approved Algorithms Not Allowed in the Approved Mode of Operation*

## 2.7 Module Design and Components

The software block diagram below shows the module, its interfaces with the operational environment and the delimitation of its cryptographic boundary.
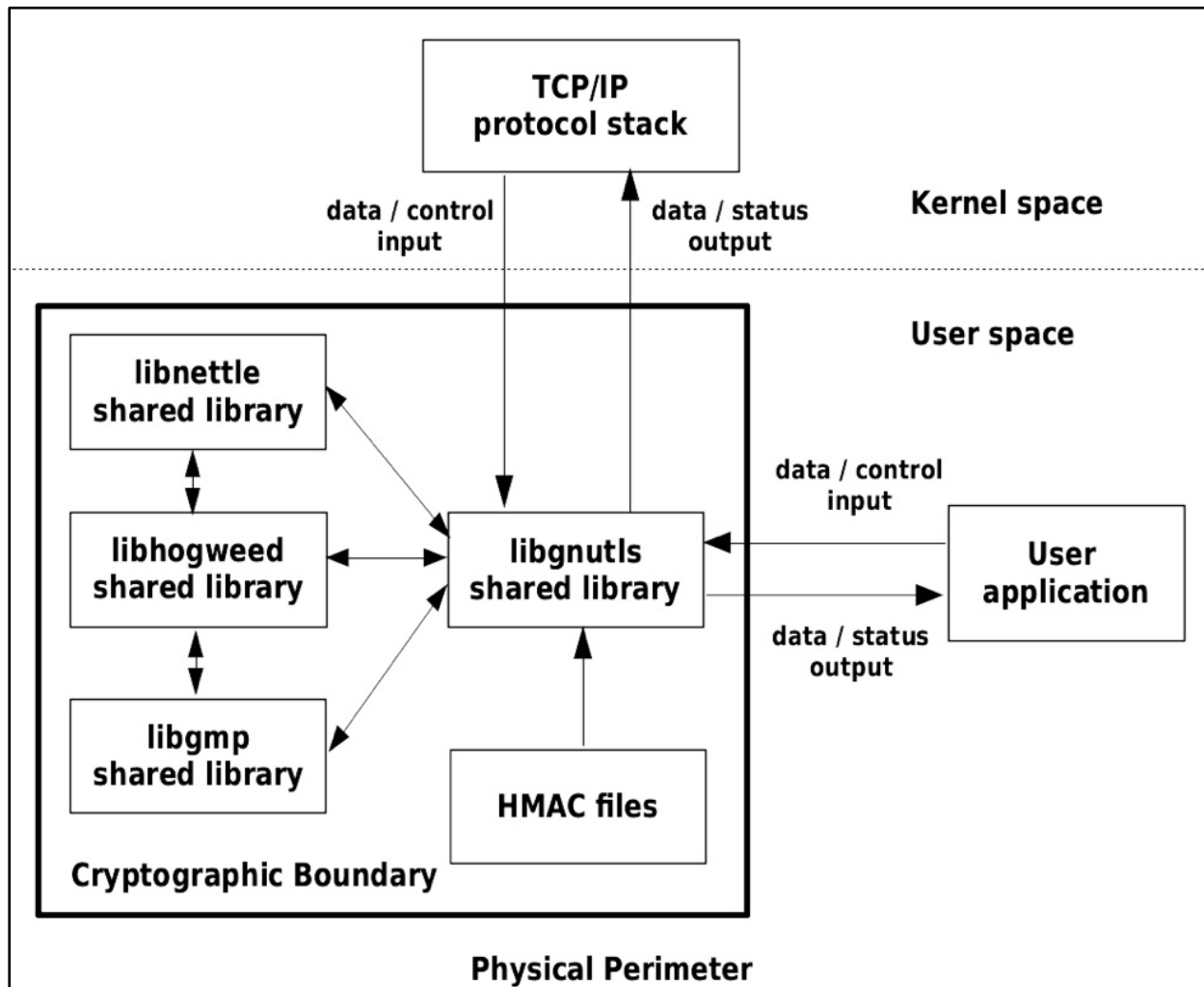
*Figure 1 – Cryptographic Boundary*

The module is implemented as a shared library. The cryptographic module boundary consists of 5 components:

- libgnutls
- libnettle
- libhogweed
- libgmp – statically linked
- gnutls hmac file

# 3  Cryptographic Module Ports and Interfaces

The logical interfaces are the API through which the applications request services. The following table summarizes the logical interfaces:

| Physical Port | Logical Interface[1] | 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 for control |
| | Status Output | API return codes, status parameters |

*Table 7 - Ports and Interfaces*

The module does not output any control data to another cryptographic module.

---

[1] The module does not implement Control Output interface.

# 4  Roles, services, and authentication

The module supports the Crypto Officer role only. This sole role is implicitly assumed by the operator of the module when performing a service. The module does not support authentication.

## 4.1 Roles

Table below describes the authorized role(s) in which the service can be performed with specification of the service input parameters and associated service output parameters.

| Role | Service | Input | Output |
|------|---------|-------|--------|
| Crypto Officer (CO) | Authenticated encryption | Key, Plaintext, IV | Ciphertext, MAC tag |
| | Authenticated decryption | Key, Ciphertext, IV, MAC tag | Plaintext |
| | Diffie-Hellman shared secret computation | Private key, public key from peer | Shared secret |
| | Digital signature generation | Message, hash algorithm, private key | Digital signature |
| | Digital signature verification | Message, signature, hash algorithm, public key | Verification result |
| | Domain parameter generation | Domain parameters input | Generated domain parameters |
| | EC Diffie-Hellman shared secret computation | Private key, public key from peer | Shared secret |
| | HKDF key derivation | Shared secret | HKDF derived key |
| | Key pair generation | RSA key size, Diffie-Hellman Safe Prime or Elliptic Curve, enabled-curve[2] | Key pair |
| | Key agreement | Private key, public key from peer | Derived key |
| | Key encapsulation | Key to be encapsulated, Key encapsulating key | Encapsulated key |
| | Key un-encapsulation | Encapsulated key, Key encapsulating key | Unencapsulated key |
| | Key wrapping | Key to be wrapped, Key wrapping key | Wrapped key |
| | Key unwrapping | Wrapped key, Key unwrapping key | Unwrapped key |
| | Message authentication code (MAC) | HMAC key or AES key, message | MAC tag |
| | Message authentication code verification | HMAC key or AES key, message, MAC tag | Pass/fail |
| | Message digest | Message | Digest of the message |
| | Password-based key derivation | Password or passphrase, salt, iteration count | PBKDF Derived key |
| | Public key verification | Key pair | Return codes/log messages |
| | Random number generation | Number of bits | Random number |

---

[2] The enabled-curve input parameter can be adjusted relying on the crypto-policies package provided as part of the RHEL OS. The usage of crypto-policies is discouraged by the vendor. Further info can be found at the vendor's documentation.

| Role | Service | Input | Output |
|------|---------|-------|--------|
| | Self-tests | N/A | Result of self-test (pass/fail) |
| | Symmetric decryption | Key, Ciphertext | Plaintext |
| | Symmetric encryption | Key, Plaintext | Ciphertext |
| | Show module name and version | N/A | Name and version information |
| | Show status | N/A | Return codes and/or log messages |
| | TLS key derivation | TLS Pre-master secret | Derived key |
| | Transport Layer Security (TLS) network protocol | Cipher-suites, Digital Certificate, Public and Private Keys, Application Data | Return codes and/or log messages, Application data |
| | Zeroization | Context containing SSPs | N/A |

*Table 8 - Roles, Service Commands, Input and Output*

## 4.2 Authentication

FIPS 140-3 does not require an authentication mechanism for level 1 modules. Therefore, the module does not implement an authentication mechanism for Crypto Officer. The Crypto Officer role is authorized to access all services provided by the module (see Table - Approved Services and Table - Non-Approved Services below).

## 4.3 Services

The table below lists all approved services that can be used in the approved mode of operation.

The following convention is used to specify access rights to an SSP:

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

The service indicator is invoked by calling the function "gnutls_fips140_get_operation_state()" and it returns "GNUTLS_FIPS140_OP_APPROVED" depending on whether the API invoked corresponds to an approved algorithm.

| Service | Description | Approved Security Functions | Keys and/or SSPs | Roles | Access rights to Keys and/or SSPs | Indicator |
|---------|-------------|-----------------------------|------------------|-------|-----------------------------------|-----------|
| **Cryptographic Services** | | | | | | |

| Service | Description | Approved Security Functions | Keys and/or SSPs | Roles | Access rights to Keys and/or SSPs | Indicator |
|---|---|---|---|---|---|---|
| Symmetric encryption | Perform AES encryption | AES-CBC AES-ECB AES-CCM AES-CFB8 AES-CMAC AES-GMAC AES-XTS | AES key | CO | W, E | GNUTLS_FIPS140 _OP_APPROVED |
| Symmetric decryption | Perform AES decryption | AES-CBC AES-ECB AES-CCM AES-GCM AES-CFB8 AES-CMAC AES-GMAC AES-XTS | AES key | | W, E | GNUTLS_FIPS140 _OP_APPROVED |
| Authenticated encryption | Encrypt a plaintext | AES-CCM | AES key | | W, E | GNUTLS_FIPS140 _OP_APPROVED |
| Authenticated decryption | Decrypt a ciphertext | AES-CCM | AES key | | W, E | GNUTLS_FIPS140 _OP_APPROVED |
| Key wrapping | Key wrapping (as part of the cipher suites in the TLS protocol) | AES-CCM AES-GCM | AES key | | W, E | GNUTLS_FIPS140 _OP_APPROVED |
| | | AES-CBC, HMAC | AES key, HMAC key | | W, E | |
| Key unwrapping | Key unwrapping (as part of the cipher suites in the TLS protocol) | AES-CCM AES-GCM | AES key | | W, E | GNUTLS_FIPS140 _OP_APPROVED |
| | | AES-CBC, HMAC | AES key, HMAC key | | W, E | |
| Key pair generation | Generate RSA, ECDSA/ECDH and DH key pairs | CKG DRBG ECDSA RSA Safe Primes Key generation | Module-generated RSA public key, Module generated RSA private key | | G, E, R | GNUTLS_FIPS140 _OP_APPROVED |
| | | | Module-generated ECDSA public key, Module generated ECDSA private key | | G, E, R | |
| | | | Module-generated Diffie-Hellman public key, Module-generated Diffie-Hellman private keys | | G, E, R | |
| | | | Module-generated EC Diffie-Hellman public key, Module-generated EC Diffie-Hellman private keys | | G, E, R | |
| | | | DRBG internal state (V value, key) | | W, E | |
| | | | "enabled-curve" parameter | | W, E | |

| Service | Description | Approved Security Functions | Keys and/or SSPs | Roles | Access rights to Keys and/or SSPs | Indicator |
|---|---|---|---|---|---|---|
| Digital signature generation | Generate RSA and ECDSA signature See Table 5 for SHA sizes | DRBG ECDSA SHA RSA | DRBG internal state (V value, key) | | W, E | GNUTLS_FIPS140_OP_APPROVED |
| | | | RSA private key | | | |
| | | | ECDSA private key | | | |
| Digital signature verification | Verify RSA, and ECDSA signature See Table 5 for SHA sizes | RSA ECDSA SHA | RSA public key | | W, E | GNUTLS_FIPS140_OP_APPROVED |
| | | | ECDSA public key | | | |
| Public key verification | Verify ECDSA public key | ECDSA | ECDSA public key | | W, E | GNUTLS_FIPS140_OP_APPROVED |
| Random number generation | Generate random bitstrings | DRBG | Entropy input | | W, E | GNUTLS_FIPS140_OP_APPROVED |
| | | | DRBG internal state (V value, key) | | E, G | GNUTLS_FIPS140_OP_APPROVED |
| | | | DRBG seed | | E, G | GNUTLS_FIPS140_OP_APPROVED |
| Message digest | Compute SHA hashes | SHA | None | | N/A | GNUTLS_FIPS140_OP_APPROVED |
| Message authentication code (MAC) | Compute HMAC | Compute HMAC | HMAC key | | W, E | GNUTLS_FIPS140_OP_APPROVED |
| | Compute AES-based CMAC | CMAC with AES | AES key | | | |
| | Compute AES-based GMAC | GMAC with AES | AES key | | | |
| Message authentication code verification | Verify MAC tag | HMAC or GMAC with AES or CMAC with AES | AES key or HMAC key | | W, E | GNUTLS_FIPS140_OP_APPROVED |
| Diffie-Hellman shared secret computation | Compute a shared secret | KAS-FFC-SSC | Diffie-Hellman public key, Diffie-Hellman private key | | W, E | GNUTLS_FIPS140_OP_APPROVED |
| | | | Diffie-Hellman Shared secret | | G, R | |
| EC Diffie-Hellman shared secret computation | Compute a shared secret | KAS-ECC-SSC | EC Diffie-Hellman public key, EC Diffie-Hellman private key | | W, E | GNUTLS_FIPS140_OP_APPROVED |
| | | | EC Diffie-Hellman Shared secret | | G, R | |
| TLS key derivation | Perform TLS key derivation | TLS v1.2 KDF RFC7627 | TLS pre-master secret | | W, E | GNUTLS_FIPS140_OP_APPROVED |
| | | | TLS master secret | | E, G | |
| | | | TLS derived key | | G, R | |
| HKDF key derivation | Perform key derivation using HKDF (in the context of TLS 1.3) | KDA HKDF | Diffie-Hellman shared secret or EC Diffie-Hellman shared secret | | W, E | GNUTLS_FIPS140_OP_APPROVED |

| Service | Description | Approved Security Functions | Keys and/or SSPs | Roles | Access rights to Keys and/or SSPs | Indicator |
|---|---|---|---|---|---|---|
| | | | HKDF derived key | | G, R | |
| Password-based key derivation | Perform password-based key derivation | PBKDF | Password/passphrase | | W, E | GNUTLS_FIPS140_OP_APPROVED |
| | | | PBKDF derived key | | G, R | |
| **Network Protocol Service** | | | | | | |
| Transport Layer Security (TLS) network protocol | Establish TLS session | Supported cipher suites in FIPS-validated configuration (see Appendix A for the complete list of valid cipher suites) | RSA public key, RSA private key, ECDSA public key, ECDSA private key | CO | W, E | GNUTLS_FIPS140_OP_APPROVED |
| | | | TLS pre-master secret, TLS master secret, Diffie Hellman private key, Diffie-Hellman public key, EC Diffie Hellman public key, EC Diffie-Hellman private key, TLS derived key, HKDF derived key | | W, E, G | |
| **Other FIPS-Related Services** | | | | | | |
| Show status | Show module status | N/A | None | CO | N/A | N/A |
| Self-tests | Perform self-tests | AES, Diffie-Hellman, EC Diffie-Hellman, ECDSA, DRBG, HMAC, RSA, SHS, HKDF, PBKDF, TLS v1.2 KDF RFC7627 | None | | N/A | N/A |
| Show module name and version | Show module name and version | N/A | None | | N/A | N/A |
| Zeroization | Zeroize SSPs | N/A | Any SSPs | | All SSPs: Z | N/A |

*Table 9 - Approved Services*

The table below lists all non-Approved services that can only be used in the non-Approved mode of operation.

| Service | Description | Algorithms Accessed | Role |
|---|---|---|---|
| **Cryptographic Services** | | | |
| Symmetric encryption | Compute the cipher for encryption | AES GCM not in the context of the TLS protocol<br>Blowfish<br>Camellia<br>CAST<br>ChaCha20<br>DES<br>GOST | CO |

| Service | Description | Algorithms Accessed | Role |
|---|---|---|---|
| **Cryptographic Services** | | | |
| | | RC2, RC4<br>Salsa20<br>Serpent<br>Triple-DES<br>Twofish | |
| Symmetric decryption | Compute the cipher for decryption | AES GCM not in the context of the TLS protocol<br>Blowfish<br>Camellia<br>CAST<br>ChaCha20<br>DES<br>GOST<br>RC2, RC4<br>Salsa20<br>Serpent<br>Triple-DES<br>Twofish | |
| Key pair generation | Generate RSA, DSA, and ECDSA key pairs | DSA<br>ECDSA with curves not listed in Table 5<br>RSA with keys smaller than 2048 bits or greater than 4096 bits | |
| Digital signature generation | Sign RSA, DSA, and ECDSA signatures | DSA<br>ECDSA with curves not listed in Table 5<br>RSA with keys smaller than 2048 bits or greater than 4096 bits | |
| Digital signature verification | Verify RSA, DSA, and ECDSA signatures | DSA<br>ECDSA with curves not listed in Table 5<br>RSA with keys smaller than 1024 bits or greater than 4096 bits | |
| Domain parameter generation | Generate domain parameter | DSA | |
| Message digest | Compute message digest | GOST<br>MD2, MD4, MD5<br>RMD160<br>SM3<br>STREEBOG | |
| Message authentication code (MAC) | Compute HMAC | HMAC with keys smaller than 112-bit<br>HMAC with GOST<br>MD2, MD4, MD5<br>RMD160<br>STREEBOG<br>UMAC | |
| Key agreement | Perform key agreement | Diffie-Hellman with keys generated with domain parameters other than safe primes<br>EC Diffie-Hellman with curves not listed in Table 5 | |
| Key encapsulation | Perform RSA key encapsulation | RSA encryption and decryption with any key sizes | |

| Service | Description | Algorithms Accessed | Role |
|---------|-------------|---------------------|------|
| **Cryptographic Services** | | | |
| Key un-encapsulation | Perform RSA key un-encapsulation | RSA encryption and decryption with any key sizes | |
| Diffie-Hellman shared secret computation | Perform DH shared secret computation | Diffie-Hellman with keys generated with domain parameters other than safe primes | |
| EC Diffie-Hellman shared secret computation | Perform ECDH shared secret computation | EC Diffie-Hellman with curves not listed in Table 5 | |
| Password-based key derivation | Perform password-based key derivation | PBKDF using non-approved message digest algorithms | |
| Public key verification | Verify ECDSA public key | ECDSA with curves not listed in Table 5 | |
| Transport Layer Security (TLS) network protocol | Establish non-supported TLS channel | Non-supported cipher suite (see Appendix A for the complete list of valid cipher) | |

*Table 10 - Non-Approved Services*

# 5  Software/Firmware security

## 5.1 Integrity Techniques

The integrity of the module is verified by comparing an HMAC-SHA2-256 value calculated at run time with the HMAC value stored in the .hmac file that was computed at build time for each software component of the module listed in section 2. The .hmac file has HMAC value for libgnutls, libnettle and libhogweed listed in section 2. If the HMAC values do not match, the test fails, and the module enters the error state.

Integrity tests are performed as part of the Pre-Operational Self-Tests.

## 5.2 On-Demand Integrity Test

The module provides the Self-Test service to perform self-tests on demand which includes the pre-operational test (i.e., integrity test) and the cryptographic algorithm self-tests (CASTs). The Self-Tests service can be called on demand by invoking the gnutls_fips140_run_self_tests() function which will perform integrity tests and the cryptographic algorithms self-tests. Additionally, the Self-Test service can be invoked by powering-off and reloading the module. During the execution of the on-demand self-tests, services are not available, and no data output is possible.

## 5.3  Executable Code

The module consists of executable code in the form of libgnutls, libnettle, libgmp and libhogweed shared libraries as stated in section 2.

# 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 2.3.

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 binaries are 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 (N/A).

# 8  Non-invasive Security

This module does not implement any non-invasive security mechanism and therefore this section is Not Applicable (N/A).

# 9  Sensitive Security Parameters Management

Table 11 summarizes the SSPs that are used by the cryptographic services implemented in the module.

| Key / SSP Name / Type | Strength | Security Function and Cert. Number | Generation | Import/Export | Establishment | Storage | Zeroization | Use & related keys |
|---|---|---|---|---|---|---|---|---|
| AES key | AES-XTS: 128, 256 bits; Other modes: 128, 192, 256 bits | AES-CBC, AES-CCM, AES-CFB8, AES-CMAC, AES-ECB AES-GCM, AES-GMAC, AES-XTS, Certs. #A3472, #A3473, #A3475, #A3476, #A3478, #A3479, #A3481, #A3550, #A3551 | N/A | MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None | N/A | RAM | gnutls_cipher_deinit() gnutls_aead_cipher_deinit() | **Use:** Symmetric encryption; Symmetric decryption; Message authentication code (MAC); Message authentication code verification; Authenticated encryption; Authenticated decryption; Key wrapping; Key unwrapping **Related SSPs:** N/A |
| HMAC key | 112–256 bits | HMAC Certs. #A3473, #A3478, #A3552 | N/A | MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None | N/A | RAM | gnutls_hmac_deinit() | **Use:** Message Authentication Code (MAC); Message authentication code verification; Key wrapping; Key unwrapping **Related SSPs:** N/A |
| Module-generated RSA public key | 112 to 256 bits | DRBG, RSA: Cert. #A3478 | Generated using the FIPS 186-4 key generation method; the random value used in key generation is obtained from the SP800-90Arev1 DRBG. | MD/EE **Import:** None **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. | N/A | RAM | gnutls_privkey_deinit() gnutls_x509_privkey_deinit() gnutls_rsa_params_deinit() | **Use:** Key pair generation **Related SSPs:** DRBG internal state (V value, key); Module-generated RSA private key |
| Module-generated RSA private key | 112 to 256 bits | DRBG, RSA: Cert. #A3478 | Generated using the FIPS 186-4 key generation method; the random value used in key generation is obtained from the SP800-90Arev1 DRBG. | MD/EE **Import:** None **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. | N/A | RAM | gnutls_privkey_deinit() gnutls_x509_privkey_deinit() gnutls_rsa_params_deinit() | **Use:** Key pair generation **Related SSPs:** DRBG internal state (V value, key); Module-generated RSA public key |
| RSA public key | 112 to 256 bits | RSA Cert. #A3478 | N/A | MD/EE | N/A | RAM | gnutls_privkey_deinit() | **Use:** Digital signature verification; |

| Key / SSP Name / Type | Strength | Security Function and Cert. Number | Generation | Import/Export | Establishment | Storage | Zeroization | Use & related keys |
|---|---|---|---|---|---|---|---|---|
| | | | | **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None | | | gnutls_x509_privkey_deinit() gnutls_rsa_params_deinit() | Transport Layer Security (TLS) network protocol **Related SSPs:** RSA private key |
| RSA private key | 112 to 256 bits | RSA Cert. #A3478 | N/A | MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None | N/A | RAM | gnutls_privkey_deinit() gnutls_x509_privkey_deinit() gnutls_rsa_params_deinit() | **Use:** Digital signature generation; Transport Layer Security (TLS) network protocol **Related SSPs:** RSA public key |
| Module-generated ECDSA public key | 112, 192, 256 bits | DRBG, ECDSA: Cert. #A3478 | Generated using the FIPS 186-4 key generation method; the random value used in key generation is obtained from the SP800-90Arev1 DRBG. | MD/EE **Import:** None **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. | N/A | RAM | gnutls_privkey_deinit() gnutls_x509_privkey_deinit() gnutls_rsa_params_deinit() | **Use:** Key pair generation **Related SSPs:** DRBG internal state (V value, key); Module-generated ECDSA private key |
| Module-generated ECDSA private key | 112, 192, 256 bits | DRBG, ECDSA: Cert. #A3478 | Generated using the FIPS 186-4 key generation method; the random value used in key generation is obtained from the SP800-90Arev1 DRBG. | MD/EE **Import:** None **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. | N/A | RAM | gnutls_privkey_deinit() gnutls_x509_privkey_deinit() gnutls_rsa_params_deinit() | **Use:** Key pair generation **Related SSPs:** DRBG internal state (V value, key); Module-generated ECDSA public key |
| ECDSA public key | 128, 192, 256 bits | ECDSA Cert. #A3478 | N/A | MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None | N/A | RAM | gnutls_privkey_deinit() gnutls_x509_privkey_deinit() gnutls_rsa_params_deinit() | **Use:** Digital signature verification; Public key verification; Transport Layer Security (TLS) network protocol **Related SSPs:** DRBG internal state (V value, key); ECDSA private key |
| ECDSA private key | 128, 192, 256 bits | ECDSA Cert. #A3478 | N/A | MD/EE | N/A | RAM | gnutls_privkey_deinit() gnutls_x509_privkey_deinit() | **Use:** Digital signature generation; Public key |

| Key / SSP Name / Type | Strength | Security Function and Cert. Number | Generation | Import/Export | Establis hment | Storag e | Zeroization | Use & related keys |
|---|---|---|---|---|---|---|---|---|
| | | | | **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None | | | gnutls_rsa_par ams_deinit() | verification; Transport Layer Security (TLS) network protocol **Related SSPs:** DRBG internal state (V value, key); ECDSA public key |
| Module-generated Diffie-Hellman public key | 112-200 bits | KAS-FFC-SSC DRBG Cert. #A3478 | Generated using the SP 800-56Arev3 Safe Primes key generation method; random values are obtained from the SP800-90Arev1 DRBG. | MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None | N/A | RAM | gnutls_dh_par ams_deinit() gnutls_pk_par ams_clear() | **Use:** Key pair generation; Transport Layer Security (TLS) network protocol **Related SSPs:** Module-generated Diffie-Hellman private key; DRBG internal state (V value, key); TLS pre-master secret |
| Module-generated Diffie-Hellman private key | 112-200 bits | KAS-FFC-SSC DRBG Cert. #A3478 | Generated using the SP 800-56Arev3 Safe Primes key generation method; random values are obtained from the SP800-90Arev1 DRBG. | MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None | N/A | RAM | gnutls_dh_par ams_deinit() gnutls_pk_par ams_clear() | **Use:** Key pair generation; Transport Layer Security (TLS) network protocol **Related SSPs:** Module-generated Diffie-Hellman public key; DRBG internal state (V value, key); TLS pre-master secret |
| Diffie-Hellman public key | 112-200 bits | KAS-FFC- SSC Cert. #A3478 | N/A | MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None | N/A | RAM | gnutls_dh_par ams_deinit() gnutls_pk_par ams_clear() | **Use:** Diffie-Hellman shared secret computation; Transport Layer Security (TLS) network protocol **Related keys:** Diffie-Hellman private key; Diffie-Hellman shared secret |
| Diffie-Hellman private key | 112-200 bits | KAS-FFC- SSC Cert. #A3478 | N/A | MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None | N/A | RAM | gnutls_dh_par ams_deinit() gnutls_pk_par ams_clear() | **Use:** Diffie-Hellman shared secret computation; Transport Layer Security (TLS) network protocol **Related keys:** Diffie-Hellman public key; Diffie-Hellman shared secret |

| Key / SSP Name / Type | Strength | Security Function and Cert. Number | Generation | Import/Export | Establishment | Storage | Zeroization | Use & related keys |
|---|---|---|---|---|---|---|---|---|
| Module-generated EC Diffie-Hellman public key | 128, 192, 256 bits | KAS-ECC-SSC DRBG Cert. #A3478 | Generated internally by the module using the ECDSA key generation method compliant with [FIPS186-4] and [SP800-56Arev3]; the random value used in key generation is obtained from the SP800-90Arev1 DRBG | MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None | N/A | RAM | gnutls_pk_params_clear() | **Use:** Key pair generation; Transport Layer Security (TLS) network protocol **Related keys:** Module-generated EC Diffie-Hellman private key; DRBG internal state (V value, key); TLS pre-master secret |
| Module-generated EC Diffie-Hellman private key | 128, 192, 256 bits | KAS-ECC-SSC DRBG Cert. #A3478 | Generated internally by the module using the ECDSA key generation method compliant with [FIPS186-4] and [SP800-56Arev3]; the random value used in key generation is obtained from the SP800-90Arev1 DRBG | MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None | N/A | RAM | gnutls_pk_params_clear() | **Use:** Key pair generation; Transport Layer Security (TLS) network protocol **Related keys:** Module-generated EC Diffie-Hellman public key; DRBG internal state (V value, key); TLS pre-master secret |
| EC Diffie-Hellman public key | 128, 192, 256 bits | KAS-ECC-SSC Cert. #A3478 | N/A | MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None | N/A | RAM | gnutls_pk_params_clear() | **Use:** EC Diffie-Hellman shared secret computation; Transport Layer Security (TLS) network protocol **Related keys:** EC Diffie-Hellman private key; EC Diffie-Hellman shared secret |
| EC Diffie-Hellman private key | 128, 192, 256 bits | KAS-ECC-SSC Cert. #A3478 | N/A | MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None | N/A | RAM | gnutls_pk_params_clear() | **Use:** EC Diffie-Hellman shared secret computation; Transport Layer Security (TLS) network protocol **Related keys:** EC Diffie-Hellman public key; EC Diffie-Hellman shared secret |
| Diffie-Hellman shared secret | 112 to 200 bits | KAS-FFC-SSC Cert. #A3478 | N/A | MD/EE | Generated during the Diffie- | RAM | zeroize_key() | **Use:** Diffie-Hellman shared secret computation; |

| Key / SSP Name / Type | Strength | Security Function and Cert. Number | Generation | Import/Export | Establishment | Storage | Zeroization | Use & related keys |
|---|---|---|---|---|---|---|---|---|
| | | | | **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. | Hellman key agreement and shared secret computation per SP800-56Arev3. | | | HKDF key derivation **Related keys:** Diffie-Hellman public key; Diffie-Hellman private key |
| EC Diffie-Hellman shared secret | 112 to 256 bits | KAS-ECC-SSC Cert. #A3478 | N/A | MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. | Generated during the EC Diffie-Hellman key agreement and shared secret computation per SP800-56Arev3. | RAM | zeroize_key() | **Use:** EC Diffie-Hellman shared secret computation; HKDF key derivation **Related keys:** EC Diffie-Hellman public key; EC Diffie-Hellman private key |
| PBKDF password or passphrase | Password strength $10^{14}$ - $10^{128}$ | PBKDF Cert. #A3478 | N/A (key material is entered via API parameters) | MD/EE **Import:** CM to TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None | N/A | RAM | Internal PBKDF state is zeroized automatically when function returns. | **Use:** Password-based key derivation **Related keys:** PBKDF derived key |
| PBKDF derived key | 112-256 bits | PBKDF Cert. #A3478 | Derived during the PBKDF | MD/EE **Import:** None **Export:** CM from TOEPP Path. Passed from the module via API parameters in plaintext (P) format. | N/A | RAM | zeroize_key() | **Use:** Password-based key derivation **Related keys:** PBKDF password or passphrase |
| HKDF derived key | 112 to 256 bits | KDA HKDF Cert. #A3477 | Derived (as part of TLSv1.3) with KDA HKDF | MD/EE **Import:** None **Export:** CM from TOEPP Path. | N/A | RAM | gnutls_deinit() | **Use:** HKDF key derivation; Transport Layer Security (TLS) network protocol |

| Key / SSP Name / Type | Strength | Security Function and Cert. Number | Generation | Import/Export | Establishment | Storage | Zeroization | Use & related keys |
|---|---|---|---|---|---|---|---|---|
| | | | | Passed from the module via API parameters in plaintext (P) format. | | | | **Related keys:** Diffie-Hellman shared secret, EC Diffie-Hellman shared secret |
| Entropy input<br><br>IG D.L compliant | 112 to 337 bits | DRBG Cert. #A3478<br><br>ESV Cert. #E47 | Obtained from the SP 800-90B compliant Non-Physical Entropy Source | **Import:** None<br>**Export:** None it remains within the cryptographic boundary. | N/A | RAM | gnutls_global_deinit() | **Use:** Random number generation<br>**Related keys:** DRBG seed |
| DRBG internal state (V value, key)<br><br>IG D.L compliant | 128 to 256 bits | DRBG Cert. #A3478 | Generated from the DRBG seed as defined in SP800-90Arev1 | **Import:** None<br>**Export:** None | N/A | RAM | gnutls_global_deinit() | **Use:** Random number generation<br>**Related keys:** DRBG seed, Module-generated ECDSA public key, Module-generated ECDSA private key, Module-generated RSA public key, Module-generated RSA private key, Module-generated Diffie-Hellman public key, Module-generated Diffie-Hellman private key, Module-generated EC Diffie-Hellman public key, Module-generated EC Diffie-Hellman private key |
| DRBG seed<br><br>IG D.L compliant | 128 to 256 bits | DRBG Cert. #A3478<br><br>ESV Cert. #E47 | Derived from entropy input as defined in SP800-90Arev1 | **Import:** None<br>**Export:** None it remains within the cryptographic boundary. | N/A | RAM | gnutls_global_deinit() | **Use:** Random number generation<br>**Related keys:** Entropy input; DRBG internal state (V value, key) |
| TLS pre-master secret | DH 112 to 256 bits ECDH 112 to 256 bits | TLS v1.2 KDF RFC7627 Certs. #A3478 | N/A | MD/EE<br><br>**Import:** CM to TOEPP Path. Passed to the module via API parameters in plaintext (P) format.<br>**Export:** None | Key agreement for Diffie-Hellman or EC Diffie-Hellman and shared secret computat | RAM | gnutls_deinit() | **Use:** TLS key derivation, Transport Layer Security (TLS) network protocol<br>**Related keys:** TLS master secret |

| Key / SSP Name / Type | Strength | Security Function and Cert. Number | Generation | Import/Export | Establis hment | Storag e | Zeroization | Use & related keys |
|---|---|---|---|---|---|---|---|---|
| | | | | | ion per SP800-56Arev3 | | | |
| TLS master secret | 112 to 256 bits | TLS v1.2 KDF RFC7627 Certs. #A3478 | Derived from TLS pre-master secret using TLS v1.2 KDF RFC7627per SP800-135rev1. | MD/EE **Import:** None **Export:** None | N/A | RAM | gnutls_deinit() | **Use:** TLS key derivation, Transport Layer Security (TLS) network protocol **Related keys:** TLS pre-master secret, TLS derived key |
| TLS derived key | 112 to 256 bits | TLS v1.2 KDF RFC7627 Certs. #A3478 | Derived from TLS master secret using TLS v1.2 KDF RFC7627 per SP800-135rev1. | MD/EE **Import:** None **Export:** CM from TOEPP Path. Passed from the module via API parameters in plaintext (P) format. | N/A | RAM | gnutls_deinit() | **Use:** TLS key derivation, Transport Layer Security (TLS) network protocol **Related keys:** TLS pre-master secret, TLS master secret |

*Table 11 - SSPs*

## 9.1 Random bit Generator

The module employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90ARev1] for the generation of random value used in asymmetric keys, and for providing a RNG service to calling applications. The approved DRBG provided by the module is the CTR_DRBG with AES-256. The DRBG does not employ prediction resistance or a derivation function. The module uses an SP800-90B-compliant Entropy Source specified in the table below to seed the DRBG.

| Entropy Source | Minimum number of bits of entropy | Details |
|---|---|---|
| SP 800-90B compliant Non-Physical Entropy Source (ESV cert. E47) | 225 bits of entropy in the 256-bit output | Userspace CPU Jitter 2.2.0 entropy source with LFSR as the non-vetted conditioning component is located within the physical perimeter of the module but 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

In accordance with FIPS 140-3 IG D.H, the cryptographic module performs Cryptographic Key Generation (CKG) for asymmetric keys according to section 5.1 and 5.2 of [SP800-133rev2] according to section 6.1 of [SP800-133rev2] (vendor affirmed) by obtaining a random bit string directly from an approved [SP800-90Arev1] DRBG and that can support the required security strength requested by the caller (without any V, as described in Additional Comments 2 of IG D.H).

- For generating RSA and ECDSA keys, the module implements asymmetric cryptographic key generation (CKG) services compliant with [FIPS186-4].

- The public and private keys used in the EC Diffie-Hellman key agreement schemes are generated internally by the module using the ECDSA key generation method compliant with [FIPS186-4] and [SP800-56Arev3].

- The public and private keys used in the Diffie-Hellman key agreement scheme are also compliant with [SP800-56Arev3]. The module generates keys using safe primes defined in RFC7919 and RFC3526, as described in the next section.

The module provides the following SSP generation methods with associated SSP sizes and strengths:

- RSA [FIPS186-4] B.3.2 Random Provable Primes - 2048, 3072, 4096-bit keys with 112-149 bits of key strength

- ECDH/ECDSA [FIPS186-4] B.4.2 Testing Candidates - P-256, P-384, P-521 elliptic curves with 128-256 bits of key strength

- Safe Primes Key Generation [SP800-56Arev3] - 2048, 3072, 4096, 6144, 8192-bit keys with 112-200 bits of key strength

The module supports the following key derivation methods according to [SP800-135rev1]:

- KDF for the TLS protocol, used as pseudo-random functions (PRF) for TLSv1.2 (RFC7627)

The module supports the following key derivation methods according to [SP800-56Cr1]:

- HKDF for the TLS protocol TLSv1.3.

The module also supports password-based key derivation (PBKDF). The implementation is compliant with option 1a of [SP800-132]. Keys derived from passwords or passphrases using this method can only be used in storage applications

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

## 9.3 SSP entry and output

SSPs are provided to the module via API input parameters in plaintext form and output via API output parameters in plaintext form within the physical perimeter of the operational environment. This is allowed by [FIPS140-3_IG] IG 9.5.A, according to the "CM Software to/from App via TOEPP Path" entry on the Key Establishment Table. The module does not support entry or output of cryptographically protected SSPs.

## 9.4 SSP establishment

The module provides Diffie-Hellman and EC Diffie-Hellman shared secret computation compliant with SP800- 56Arev3, in accordance with scenario 2 (1) of IG D.F and used as part of the TLS protocol key exchange in accordance with scenario 2 (2) of IG D.F; that is, the shared secret computation (KAS-FFC-SSC and KAS-ECC-SSC) followed by the derivation of the keying material using SP800-135rev1 KDF and SP800-56Crev1 KDF.

For Diffie-Hellman, the module supports the use of safe primes from RFC7919 for domain parameters and key generation, which are used in the TLS key agreement implemented by the module.

- TLS (RFC7919)

  ◦ ffdhe2048 (ID = 256)

  ◦ ffdhe3072 (ID = 257)

◦  ffdhe4096 (ID = 258)

◦  ffdhe6144 (ID = 259)

◦  ffdhe8192 (ID = 260)

The module also supports the use of safe primes from RFC3526, which are part of the Modular Exponential (MODP) Diffie-Hellman groups that can be used for Internet Key Exchange (IKE). Note that the module only implements key generation and verification, and shared secret computation using safe primes, but no part of the IKE protocol.

• IKEv2 (RFC3526)

◦  MODP-2048 (ID=14)

◦  MODP-3072 (ID=15)

◦  MODP-4096 (ID=16)

◦  MODP-6144 (ID=17)

◦  MODP-8192 (ID=18)

The module also provides the following key transport mechanisms:

• Key wrapping using AES-CCM, AES-GCM, and AES-CBC with HMAC, used in the context of the TLS protocol cipher suites (in compliance with IG D.G).

According to Table 2: Comparable strengths in [SP 800-57rev5], the key sizes of AES, Diffie-Hellman and EC Diffie-Hellman provides the following security strength in Approved mode of operation:

• AES key wrapping using AES-CCM, AES-GCM, and AES in CBC mode and HMAC, provides between 128 or 256 bits of encryption strength.

• Diffie-Hellman shared secret computation provides between 112 and 200 bits of security strength.

• EC Diffie-Hellman shared secret computation provides between 128 and 256 bits of security strength.

## 9.5 SSP storage

Symmetric keys, public and private keys are provided to the module by the calling application via API input parameters and are destroyed by the module when invoking the appropriate API function calls.

The module does not perform persistent storage of SSPs. The SSPs are temporarily stored in the RAM in plaintext form. SSPs are provided to the module by the calling process and are destroyed when released by the appropriate zeroization function calls.

## 9.6 SSP Zeroization

The memory occupied by SSPs is allocated by regular memory allocation operating system calls. The application that is acting as the CO is responsible for calling the appropriate zeroization functions provided in the module's API and listed in Table 11. Calling the gnutls_deinit() will zeroize the SSPs stored in the TLS protocol internal state and also invoke the corresponding API functions listed in Table 11 to zeroize SSPs. The zeroization functions overwrite the memory occupied by SSPs with "zeros" and deallocate the memory with the regular memory deallocation operating system call. The completion of a zeroization routine(s) will indicate that a zeroization procedure succeeded. All data output is inhibited during zeroization.

# 10 Self-tests

The module performs the pre-operational self-test and CASTs automatically when the module is loaded into memory. Pre-operational self-test ensure that the module is not corrupted, and the CASTs ensure that the cryptographic algorithms work as expected. While the module is executing the self-tests, the module services are not available, and input and output are inhibited. The module is not available for use by the calling application until the pre-operational self-test and the CASTs are completed successfully. After the pre-operational test and the CASTs succeed, the module becomes operational. If any of the pre-operational test or any of the CASTs fail an error message is returned, and the module transitions to the error state.

## 10.1 Pre-operational Software Integrity Test

The module performs the following pre-operational tests: the integrity test of the shared libraries that comprise the module using HMAC-SHA2-256. The details of integrity test are provided in section 5.1. Prior the first use, a CAST is executed for the algorithms used in the Pre-operational Self-Tests.

## 10.2 Conditional Self-Tests

The following sub-sections describe the conditional self-tests supported by the module. If one of the conditional self-tests fail, the module transitions to the 'Error' state and a corresponding error indication is given.

The entropy source performs its required self-tests; those are not listed here, as the entropy source is not part of the cryptographic boundary of the module.

## 10.2.1    Conditional Cryptographic algorithm tests

The module performs cryptographic algorithm self-tests (CASTs) on all approved cryptographic algorithms. The CASTs consist of Known Answer Tests for all the approved cryptographic algorithms.

| Algorithm | Test |
|---|---|
| AES | KAT AES CBC mode with 128-bit and 256-bit keys, encryption and decryption (separately tested) <br> KAT AES CFB8 mode with 256-bit key, encryption and decryption (separately tested) <br> KAT AES GCM mode with 256-bit key, encryption and decryption (separately tested) <br> KAT AES XTS mode with 256-bit keys, encryption and decryption (separately tested) <br> KAT AES-CMAC with 256-bit key size MAC generation |
| Diffie-Hellman | Primitive "Z" Computation KAT with ffdhe3072 |
| DRBG | KAT CTR_DRBG with AES with 256-bit keys without DF, without PR |
| DRBG | Health tests according to section 11.3 of [SP800-90Arev1] |
| EC Diffie-Hellman | Primitive "Z" Computation KAT with P-256 curve |
| ECDSA | KAT ECDSA with P-256 using SHA-256, P-384 using SHA-384, and P-521 using SHA-512, signature generation and verification (separately tested) |
| HKDF KDA | KAT with SHA-256 |
| HMAC | KAT HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512 |
| PBKDF KDF | KAT with SHA-256 with 4096 iterations and 288-bit salt |
| RSA | KAT RSA PKCS#1 v1.5 with 2048-bit key using SHA-256, signature generation and verification (separately tested) |
| SHA-3 | KAT SHA3-224, SHA3-256, SHA3-384, SHA3-512 |

| Algorithm | Test |
|---|---|
| TLS v1.2 KDF RFC7627 | KAT with SHA-256 |

*Table 13 - Conditional Cryptographic Algorithm Self-Tests*

## 10.2.2    Conditional Pairwise Consistency Test

The module performs the Pair-wise Consistency Tests (PCT) shown in the following table. If any of the tests fails, the module returns an error code and enters the Error state. When the module is in the Error state, no data is output, and cryptographic operations are not allowed.

| Algorithm | Test |
|---|---|
| ECDSA key generation | PCT using SHA-256, signature generation and verification. |
| RSA key generation | PCT using PKCS#1 v1.5 with SHA-256, signature generation and verification |
| Diffie-Hellman key generation | PCT according to section 5.6.2.1.4 of [SP800-56Arev3] |
| EC Diffie-Hellman key generation | Covered by ECDSA PCT as allowed by IG 10.3.A additional comment 1 |

*Table 14 - Pairwise Consistency Test*

## 10.2.3    Periodic/On-Demand Self-Test

The module provides the Self-Test service to perform self-tests on demand which includes the pre-operational test (i.e., integrity test) and the cryptographic algorithm self-tests (CASTs). The Self-Tests service can be called on demand by invoking the gnutls_fips140_run_self_tests() function which will perform integrity tests and the cryptographic algorithms self-tests. Additionally, the Self-Test service can be invoked by powering-off and reloading the module. During the execution of the on-demand self-tests, services are not available, and no data output is possible.

## 10.3 Error States

When the module fails any pre-operational self-test or conditional test, the module will return an error code to indicate the error and enters error state. Any further cryptographic operations and the data output via the data output interface are inhibited. The calling application can obtain the module state by calling the gnutls_fips140_get_operation_state() API function. The function returns GNUTLS_FIPS140_OP_ERROR if the module is in the Error state.

The following table shows the error codes and the corresponding condition:

| Error State | Cause of Error | Status Indicator |
|---|---|---|
| Error State | When the integrity tests or KAT fail at power-up. | GNUTLS_E_SELF_TEST_ERROR (-400) |
| | When the KAT of DRBG fails during pre-operational tests | GNUTLS_E_RANDOM_FAILED (-206) |
| | When the new generated key pair fails the PCT | GNUTLS_E_PK_GENERATION_ERROR (-403) |
| | When the module is in error state and caller requests cryptographic operations | GNUTLS_E_LIB_IN_ERROR_STATE (-402) |

*Table 15 - Error States*

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 the per-

operational self-test and the CASTs to recover from these errors. If failures persist, the module must be re-installed.

# 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 gnutls-3.7.6-19.el9_0.x86_64 RPM package for x86 systems or gnutls-3.7.6-19.el9_0.s390x RPM package for s390 systems or gnutls-3.7.6-19.el9_0.ppc64le RPM package for ppc64le systems.

### 11.1.1     End of Life Procedure

For secure sanitization of the cryptographic module, the module needs first to be powered off, which will zeroize all keys and CSPs in volatile memory. Then, for actual deprecation, the module shall be upgraded to a newer version that is FIPS 140-3 validated.

The module does not possess persistent storage of SSPs, so further sanitization steps are not needed.

## 11.2 Crypto Officer Guidance

The binaries of the 'Red Hat Enterprise Linux 9 gnutls version 3.7.6-66803fa128d6a6e5' are contained in the RPM packages for delivery listed below.

Before the 'Red Hat Enterprise Linux 9 gnutls' RPM packages are installed, the RHEL 9 system must operate in Approved mode. This can be achieved by:

- Starting the installation in Approved mode. Add 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 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 Approved mode by executing the `fips-mode-setup --check` command, which should output "FIPS mode is enabled."

The following RPM packages contain the FIPS validated module:

| Processor Architecture | RPM Packages |
|---|---|
| Intel 64-bit | gnutls- 3.7.6-19.el9_0.x86_64.rpm<br>nettle- 3.8-3.el9_0.x86_64.rpm |
| z16 64-bit | gnutls - 3.7.6-19.el9_0.s390x.rpm<br>nettle - 3.8-3.el9_0.s390x.rpm |
| POWER10 64-bit | gnutls - 3.7.6-19.el9_0.ppc64le.rpm<br>nettle - 3.8-3.el9_0.ppc64le.rpm |

*Table 16 - RPM packages*

### 11.2.1     TLS

The TLS protocol implementation provides both server and client sides. In order to operate in the approved mode, digital certificates used for server and client authentication shall comply with the restrictions of key size and message digest algorithms imposed by [SP800-131Arev2]. In addition, as required also by [SP800-131Arev2], Diffie-Hellman with keys smaller than 2048 bits must not be used.

The TLS protocol lacks the support to negotiate the used Diffie-Hellman key sizes. To ensure full support for all TLS protocol versions, the TLS client implementation of the module accepts Diffie-Hellman key sizes smaller than 2048 bits offered by the TLS server.

For complying with the requirement to not allow Diffie-Hellman key sizes smaller than 2048 bits, the Crypto Officer must ensure that:

- in case the module is used as a TLS server, the Diffie-Hellman parameters must be 2048 bits or larger;

- in case the module is used as a TLS client, the TLS server must be configured to only offer Diffie-Hellman keys of 2048 bits or larger.

## 11.2.2    AES XTS

The AES algorithm in XTS mode can be only used for the cryptographic protection of data on storage devices, as specified in [SP800-38E]. The length of a single data unit encrypted with the XTS-AES shall not exceed $2^{20}$ AES blocks, that is 16MB of data.

The module implements a check that ensures, before performing any cryptographic operation, that the two AES keys used in AES XTS mode are not identical (in compliance with IG C.I) .

Note: AES-XTS shall be used with 128 and 256-bit keys only. AES-XTS with 192-bit keys is not an Approved service.

## 11.2.3    AES GCM IV

The module implements AES GCM for being used in the TLS v1.2 and v1.3 protocols. AES GCM IV generation is in compliance with [FIPS140-3_IG] IG C.H for both protocols as follows:

- For TLS v1.2, IV generation is in compliance with scenario 1.a of IG C.H and [RFC5288]. The module supports acceptable AES-GCM ciphersuites from section 3.3.1 of [SP800-52rev2].

- For TLS v1.3, IV generation is in compliance with scenario 5 of IG C.H and [RFC8446]. The module supports acceptable AES-GCM ciphersuites from section 3.3.1 of [SP800-52rev2].

The IV generated in both scenarios is only used within the context of the TLS protocol implementation. The nonce_explicit part of the IV does not exhaust the maximum number of possible values for a given session key. The design of the TLS protocol in this module implicitly ensures that the nonce_explicit, or counter portion of the IV will not exhaust all of its possible values. In case the module's power is lost and then restored, the key used for the AES GCM encryption or decryption shall be redistributed.

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. This is in compliance with Scenario 2 of FIPS 140-3 IG C.H.

## 11.2.4    Key Derivation using SP 800-132 PBKDF

The module provides password-based key derivation (PBKDF), compliant with SP800-132 and IG D.N. The module supports option 1a from section 5.4 of [SP800-132], in which the Master Key (MK) or a segment of it is used directly as the Data Protection Key (DPK).

In accordance with [SP800-132], 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 (this is verified by the module to determine the service is approved).

- A portion of the salt, with a length of at least 128 bits (this is verified by the module to determine the service is approved), shall be generated randomly using the SP800-90Arev1 DRBG,

- 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 minimum value shall be 1000 (this is verified by the module to determine the service is approved).

- Passwords or passphrases, used as an input for the PBKDF, shall not be used as cryptographic keys.

- The length of the password or passphrase shall be of at least 14 characters (this is verified by the module to determine the service is approved), and shall consist of lower-case, upper-case and numeric characters. The probability of guessing the value is estimated to be $10^{-14}$ (assuming all digits)

The calling application shall also observe the rest of the requirements and recommendations specified in [SP800-132].

## 11.2.5    Compliance to SP 800-56ARev3 assurances

To comply with the assurances listed in section 5.6.2 of SP 800-56ARev3, the module shall be used together with an application that implements the "TLS protocol" and the following steps shall be performed.

1. The entity using the module, must use the module's "Key pair generation" service for generating DH/ECDH ephemeral keys. This meets the assurances required by key pair owner defined in the section 5.6.2.1 of SP 800-56ARev3.

2. As part of the module's shared secret computation (SSC) service, the module internally performs the public key validation on the peer's public key passed in as input to the SSC function. This meets the public key validity assurance required by the sections 5.6.2.2.1/5.6.2.2.2 of SP 800-56ARev3.

The module does not support static keys therefore the "assurance of peer's possession of private key" is not applicable.

# 12 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.

# Appendix A.  TLS Cipher Suites

The module supports the following cipher suites for the TLS protocol version 1.0, 1.1, 1.2 and 1.3, compliant with section 3.3.1 of [SP800-52rev2]. Each cipher suite defines the key exchange algorithm, the bulk encryption algorithm (including the symmetric key size) and the MAC algorithm.

| Cipher Suite | ID | Reference |
|---|---|---|
| TLS_DH_RSA_WITH_AES_128_CBC_SHA | { 0x00, 0x31 } | RFC3268 |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA | { 0x00, 0x33 } | RFC3268 |
| TLS_DH_RSA_WITH_AES_256_CBC_SHA | { 0x00, 0x37 } | RFC3268 |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA | { 0x00, 0x39 } | RFC3268 |
| TLS_DH_RSA_WITH_AES_128_CBC_SHA256 | { 0x00,0x3F } | RFC5246 |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 | { 0x00,0x67 } | RFC5246 |
| TLS_DH_RSA_WITH_AES_256_CBC_SHA256 | { 0x00,0x69 } | RFC5246 |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 | { 0x00,0x6B } | RFC5246 |
| TLS_PSK_WITH_AES_128_CBC_SHA | { 0x00, 0x8C } | RFC4279 |
| TLS_PSK_WITH_AES_256_CBC_SHA | { 0x00, 0x8D } | RFC4279 |
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 | { 0x00, 0x9E } | RFC5288 |
| TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 | { 0x00, 0x9F } | RFC5288 |
| TLS_DH_RSA_WITH_AES_128_GCM_SHA256 | { 0x00, 0xA0 } | RFC5288 |
| TLS_DH_RSA_WITH_AES_256_GCM_SHA384 | { 0x00, 0xA1 } | RFC5288 |
| TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA | { 0xC0, 0x04 } | RFC4492 |
| TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA | { 0xC0, 0x05 } | RFC4492 |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA | { 0xC0, 0x09 } | RFC4492 |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA | { 0xC0, 0x0A } | RFC4492 |
| TLS_ECDH_RSA_WITH_AES_128_CBC_SHA | { 0xC0, 0x0E } | RFC4492 |
| TLS_ECDH_RSA_WITH_AES_256_CBC_SHA | { 0xC0, 0x0F } | RFC4492 |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA | { 0xC0, 0x13 } | RFC4492 |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA | { 0xC0, 0x14 } | RFC4492 |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 | { 0xC0, 0x23 } | RFC5289 |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 | { 0xC0, 0x24 } | RFC5289 |
| TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 | { 0xC0, 0x25 } | RFC5289 |
| TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 | { 0xC0, 0x26 } | RFC5289 |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 | { 0xC0, 0x27 } | RFC5289 |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 | { 0xC0, 0x28 } | RFC5289 |

| Cipher Suite | ID | Reference |
|---|---|---|
| TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 | { 0xC0, 0x29 } | RFC5289 |
| TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 | { 0xC0, 0x2A } | RFC5289 |
| TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 | { 0xC0, 0x2B } | RFC5289 |
| TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 | { 0xC0, 0x2C } | RFC5289 |
| TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 | { 0xC0, 0x2D } | RFC5289 |
| TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384 | { 0xC0, 0x2E } | RFC5289 |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | { 0xC0, 0x2F } | RFC5289 |
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 | { 0xC0, 0x30 } | RFC5289 |
| TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 | { 0xC0, 0x31 } | RFC5289 |
| TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384 | { 0xC0, 0x32 } | RFC5289 |
| TLS_DHE_RSA_WITH_AES_128_CCM | { 0xC0, 0x9E } | RFC6655 |
| TLS_DHE_RSA_WITH_AES_256_CCM | { 0xC0, 0x9F } | RFC6655 |
| TLS_DHE_RSA_WITH_AES_128_CCM_8 | { 0xC0, 0xA2 } | RFC6655 |
| TLS_DHE_RSA_WITH_AES_256_CCM_8 | { 0xC0, 0xA3 } | RFC6655 |
| TLS_AES_128_GCM_SHA256 | { 0x13, 0x01 } | RFC8446 |
| TLS_AES_256_GCM_SHA384 | { 0x13, 0x02 } | RFC8446 |
| TLS_AES_128_CCM_SHA256 | { 0x13, 0x04 } | RFC8446 |
| TLS_AES_128_CCM_8_SHA256 | { 0x13, 0x05 } | RFC8446 |

# Appendix B.  Glossary and Abbreviations

| | |
|---|---|
| AES | Advanced Encryption Standard |
| AES-NI | Advanced Encryption Standard New Instructions |
| 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 Mode |
| DES | Data Encryption Standard |
| DF | Derivation Function |
| DSA | Digital Signature Algorithm |
| DRBG | Deterministic Random Bit Generator |
| ECB | Electronic Code Book |
| ECC | Elliptic Curve Cryptography |
| FFC | Finite Field Cryptography |
| FIPS | Federal Information Processing Standards Publication |
| GCM | Galois Counter Mode |
| GMAC | Galois Counter Mode Message Authentication Code |
| HMAC | Hash Message Authentication Code |
| KAS | Key Agreement Scheme |
| KAT | Known Answer Test |
| KW | AES Key Wrap |
| MAC | Message Authentication Code |
| NIST | National Institute of Science and Technology |
| PAA | Processor Algorithm Acceleration |
| PAI | Processor Algorithm Implementation |
| PBKDF2 | Password-based Key Derivation Function v2 |
| PKCS | Public-Key Cryptography Standards |
| PCT | Pairwise Consistency Test |
| PR | Prediction Resistance |
| RNG | Random Number Generator |
| RSA | Rivest, Shamir, Addleman |
| SHA | Secure Hash Algorithm |
| SHS | Secure Hash Standard |

# Appendix C.  References

| | |
|---|---|
| FIPS140-3 | **FIPS PUB 140-3 - Security Requirements For Cryptographic Modules**<br>March 2019<br>https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf |
| FIPS140-3_IG | **Implementation Guidance for FIPS PUB 140-3 and the Cryptographic Module Validation Program**<br>March 2024<br>https://csrc.nist.gov/csrc/media/Projects/cryptographic-module-validation-program/documents/fips%20140-3/FIPS%20140-3%20IG.pdf |
| FIPS180-4 | **Secure Hash Standard (SHS)**<br>August 2015<br>http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf |
| FIPS186-4 | **Digital Signature Standard (DSS)**<br>July 2013<br>http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf |
| FIPS197 | **Advanced Encryption Standard**<br>November 2001<br>http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf |
| FIPS198-1 | **The Keyed Hash Message Authentication Code (HMAC)**<br>July 2008<br>http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf |
| FIPS202 | **SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions**<br>August 2015<br>http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf |
| PKCS#1 | **Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1**<br>February 2003<br>http://www.ietf.org/rfc/rfc3447.txt |
| SP800-38A | **NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques**<br>December 2001<br>https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf |
| SP800-38B | **NIST Special Publication 800-38B - Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication**<br>May 2005<br>https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf |
| SP800-38C | **NIST Special Publication 800-38C - Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality**<br>July 2007<br>http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf |
| SP800-38D | **NIST Special Publication 800-38D - Recommendation for Block Cipher Modes of Operation:  Galois/Counter Mode (GCM) and GMAC**<br>November 2007<br>https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf |

| SP800-38E | **NIST Special Publication 800-38E - Recommendation for Block Cipher Modes of Operation: The XTS AES Mode for Confidentiality on Storage Devices** <br> January 2010 <br> https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38e.pdf |
|---|---|
| SP800-52rev2 | **NIST Special Publication 800-52 Revision 2 - Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations** <br> August 2019 <br> https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf |
| SP800-56ARev3 | **NIST Special Publication 800-56A Revision 3 - Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography** <br> April 2018 <br> https://doi.org/10.6028/NIST.SP.800-56Ar3 |
| SP800-56CRev2 | **Recommendation for Key Derivation through Extraction-then-Expansion** <br> August 2020 <br> https://doi.org/10.6028/NIST.SP.800-56Cr2 |
| SP800-57rev5 | **NIST Special Publication 800-57 Part 1 Revision 5 - Recommendation for Key Management Part 1: General** <br> May 2020 <br> https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf |
| SP800-90ARev1 | **NIST Special Publication 800-90A - Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** <br> June 2015 <br> http://dx.doi.org/10.6028/NIST.SP.800-90Ar1 |
| SP800-90B | **NIST Special Publication 800-90B - Recommendation for the Entropy Sources Used for Random Bit Generation** <br> January 2018 <br> https://doi.org/10.6028/NIST.SP.800-90B |
| SP800-131Arev2 | **NIST Special Publication 800-131 Revision 2 - Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths** <br> March 2019 <br> https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf |
| SP800-132 | **NIST Special Publication 800-132 - Recommendation for Password-Based Key Derivation - Part 1: Storage Applications** <br> December 2010 <br> https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf |
| SP800-133Rev2 | **NIST Special Publication 800-133 - Recommendation for Cryptographic Key Generation** <br> June 2020 <br> https://doi.org/10.6028/NIST.SP.800-133r2 |
| SP800-135rev1 | **NIST Special Publication 800-135 Revision 1 - Recommendation for Existing Application-Specific Key Derivation Functions** <br> December 2011 <br> http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf |

SP800-140B    **NIST Special Publication 800-140B - CMVP Security Policy Requirements**
March 2020
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-140B.pdf

RFC8446       **The Transport Layer Security (TLS) Protocol Version 1.3**
August 2018
https://www.ietf.org/rfc/rfc8446.txt

RFC7919       **Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)**
August 2016
https://www.ietf.org/rfc/rfc7919.txt

RFC3526       **More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)**
May 2003
https://www.ietf.org/rfc/rfc3526.txt

RFC7627       **Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension**
September 2015
https://www.ietf.org/rfc/rfc7627.txt