# Amazon Linux 2 GnuTLS Cryptographic Module

## Module Version 1.0

# FIPS 140-2 Non-Proprietary Security Policy

## Document Version 1.2

## Last update: 2020-Mar-06


Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

# Table of Contents

# List of Tables



# List of Figures

## Copyrights and Trademarks

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

# 1 Introduction

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

## 1.1 Purpose of the Security Policy

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

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

## 1.2 Target Audience

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

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

# 2 Cryptographic Module Specification

## 2.1 Module Overview

The Amazon Linux 2 GnuTLS Cryptographic Module (hereafter referred to as the “module”) is a set of libraries implementing general purpose network protocols and FIPS 140-2 Approved cryptographic algorithms. The module supports the Transport Layer Security (TLS) Protocol defined in [RFC5246] and the Datagram Transport Layer Security (DTLS) Protocol defined in [RFC4347]. 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 to provide secure communication with other peers through the network.

## 2.2 FIPS 140-2 Validation Scope

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

*Table 1: FIPS 140-2 Security Requirements.*

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

## 2.3 Definition of the Cryptographic Module

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

- gnutls-3.3.29-9.amzn2.x86_64.rpm
- nettle-2.7.1-8.amzn2.0.2.x86_64.rpm
- gmp-6.0.0-15.amzn2.0.2.x86_64.rpm

Table 2 summarizes the components of the cryptographic module.

*Table 2: Components of the module.*

| Component | Description |
|---|---|
| /usr/lib64/libgnutls.so.28.43.3 | This library provides the main interface that allows the calling applications to request |

| Component | Description |
|---|---|
| | cryptographic services. The cryptographic algorithm implementations provided by this library include the TLS protocol, DRBG, RSA key generation, Diffie-Hellman and EC Diffie-Hellman. |
| /usr/lib64/libnettle.so.4.7 | This library provides cryptographic algorithm implementations such as AES, Triple-DES, SHA2, HMAC, RSA Digital Signature, DSA and ECDSA. |
| /usr/lib64/libhogweed.so.2.5 | This library includes the primitives used by libgnutls and libnettle to support the asymmetric cryptographic operations. |
| /usr/lib64/libgmp.so.10.2.0 | This library provides the big number arithmetic operations to support the asymmetric cryptographic operations. |
| /usr/lib64/.libgnutls.so.28.43.3.hmac | HMAC-SHA2-256 value of the associated library for integrity check during the power-on. |
| /usr/lib64/.libnettle.so.4.7.hmac | HMAC-SHA2-256 value of the associated library for integrity check during the power-on. |
| /usr/lib64/.libhogweed.so.2.5.hmac | HMAC-SHA2-256 value of the associated library for integrity check during the power-on. |
| /usr/lib64/fipscheck/libgmp.so.10.2.0.hmac | HMAC-SHA2-256 value of the associated library for integrity check during the power-on. |

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

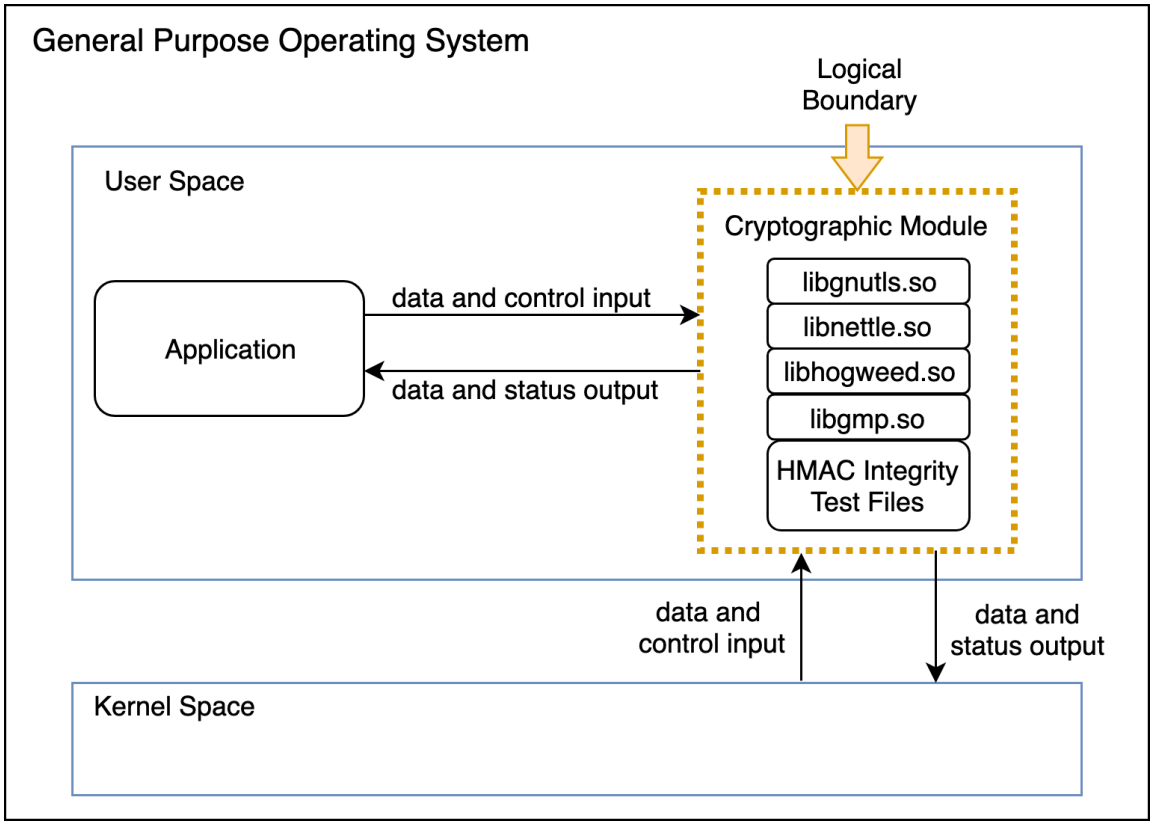


*Figure 1: Logical cryptographic boundary.*

## 2.4 Definition of the Physical Cryptographic Boundary

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

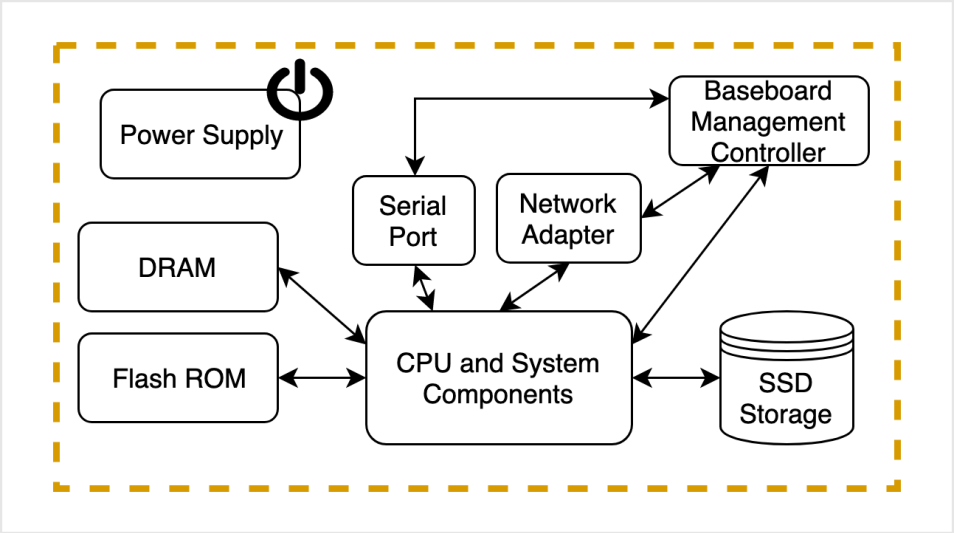


*Figure 2: Hardware block diagram.*

## 2.5 Tested Operational Environments

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

*Table 3: Tested operational environments.*

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

## 2.6 Modes of Operation

The module supports two modes of operation.

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

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

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

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

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

# 3 Module Ports and Interfaces

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

The logical interfaces are the API through which applications request services and receive output data through return values or modified data referenced by pointers. Table 4 summarizes the logical interfaces and the power input.

*Table 4: Ports and interfaces.*

| Logical Interface | Description |
|---|---|
| Data Input | API input parameters for data. |
| Data Output | API output parameters for data. |
| Control Input | API function calls. |
| Status Output | API return codes, error message. |

# 4 Roles, Services and Authentication

## 4.1 Roles

The module supports the following roles:

- **User role**: performs all services (in both FIPS mode and non-FIPS mode of operation), except module installation and configuration. This role is assumed by the calling application accessing the module.
- **Crypto Officer role**: performs module installation and configuration.

The User and Crypto Officer roles are implicitly assumed depending on the service requested.

## 4.2 Services

The module provides services to calling applications that assume the user role, and human users assuming the Crypto Officer role. Table 5 and Table 6 depict all services, which are described with more detail in the user documentation.

The tables use the following convention when specifying the access permissions that the module has for each CSP or key.

- **Create (C)**: the calling application can create a new CSP.
- **Read (R)**: the calling application can read the CSP.
- **Update (U)**: the calling application can write a new value to the CSP.
- **Zeroize (Z)**: the calling application can zeroize the CSP.
- **N/A**: the calling application does not access any CSP or key during its operation.

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

### 4.2.1 Services in the FIPS-Approved Mode of Operation

Table 5 provides a full description of FIPS Approved services and the non-Approved but Allowed services provided by the module in the FIPS-approved mode of operation and lists the roles allowed to invoke each service.

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

| Service | Service Description and Algorithms | Role | | Keys and CSPs | Access Types |
|---|---|---|---|---|---|
| | | U | CO | | |
| Symmetric Encryption/Decryption | Encrypts or decrypts a block of data using AES or Triple-DES. | ✓ | | AES or Triple-DES Key | R |
| RSA Key Generation | Generate RSA asymmetric keys. | ✓ | | RSA public/private keys | C, R, U |
| DSA Key Generation | Generate DSA asymmetric keys. | ✓ | | DSA public/private keys | C, R, U |
| ECDSA Key Generation | Generate ECDSA asymmetric keys. | ✓ | | ECDSA public/private keys | C, R, U |
| RSA Digital Signature Generation and Verification | Sign and verify signature operations for RSA PCKS#1v1.5 and X9.31. | ✓ | | RSA public/private keys | R |

| Service | Service Description and Algorithms | Role | | Keys and CSPs | Access Types |
|---|---|---|---|---|---|
| | | U | CO | | |
| DSA Digital Signature Generation and Verification | Sign and verify signature operations for DSA. | ✓ | | DSA public/private keys | R |
| ECDSA Digital Signature Generation and Verification | Sign and verify signature for ECDSA. | ✓ | | ECDSA public/private keys | R |
| RSA Public Key Verification | Verify validity of an RSA public key. | ✓ | | RSA public key | R |
| DSA Public Key Verification | Verify validity of a DSA public key. | ✓ | | DSA public key | R |
| ECDSA Public Key Verification | Verify validity of an ECDSA public key. | ✓ | | ECDSA public key | R |
| TLS Network Protocol v1.0, v1.1, v1.2 | Provide data encryption and authentication over TLS network protocol.<br>Ciphersuites composed exclusively of algorithms listed in Table 7 or Table 8. | ✓ | | AES key<br>Triple-DES key<br>HMAC key<br>Pre-master secret<br>Master secret<br>Diffie-Hellman public/private key pair<br>EC Diffie-Hellman public/private key pair<br>RSA, DSA, ECDSA public/private keys | C, R, U |
| TLS Key Derivation | Establish a TLS secure channel.<br>KDF in TLS v1.0/1.1, TLS v1.2. | ✓ | | Pre-master secret, master secret, derived keys (AES, Triple-DES, HMAC), KDF internal state | C, R, U |
| Diffie-Hellman Key Agreement | Establish a shared secret.<br>KAS FFC | ✓ | | Diffie-Hellman public/private keys | C, R, U |
| EC Diffie-Hellman Key Agreement | Establish a shared secret.<br>KAS ECC | ✓ | | EC Diffie-Hellman public/private keys | C, R, U |
| Key Wrapping | Wrapping/unwrapping a key using RSA encrypt/decrypt primitives. | ✓ | | RSA public/private keys | R |
| Certificate Management | X.509 certificate handling (digital signature, key and certificate import and export). | ✓ | | RSA, DSA, and ECDSA public/private keys associated to an X.509 certificate | R, U |
| | Authenticate and verify authentication of data using HMAC-SHA-1, HMAC-SHA2- | ✓ | | HMAC Key | R |

| Service | Service Description and Algorithms | Role | | Keys and CSPs | Access Types |
|---|---|---|---|---|---|
| | | U | CO | | |
| Message Authentication Code (MAC) | 224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512. | | | | |
| | AES-GMAC with AES-128, AES-256 | ✓ | | AES key | R |
| Message Digest | Hash a block of data with SHS (SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512). | ✓ | | None | N/A |
| Random Number Generation | Generate random numbers based on the SP 800-90A DRBG. | ✓ | | Entropy input string and internal state | C, R, U |
| **Other FIPS-related Services** | | | | | |
| Show Status | Show status of the module state | ✓ | | None | N/A |
| Self-Test | Initiate power-on self-tests | ✓ | | None | N/A |
| Zeroization | Zeroize all critical security parameters | ✓ | | All keys and CSPs | Z |
| Module Installation | Installation of the module | | ✓ | None | N/A |
| Module Configuration | Configuration of the module | | ✓ | None | N/A |

### 4.2.2 Services in the Non-FIPS-Approved Mode of Operation

Table 6 presents the services only available in non-FIPS-approved mode of operation.

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

| Service | Service Description and Algorithms | Role | | Keys | Access Types |
|---|---|---|---|---|---|
| | | U | CO | | |
| RSA key wrapping | Encrypt or decrypt using RSA key sizes not listed in Table 7 or Table 8. | ✓ | | RSA key pair | R |
| Symmetric Encryption/Decryption | Encrypt or decrypt using symmetric algorithms not listed in Table 7. | ✓ | | Blowfish, Camellia, CAST-128, DES, RC2, RC4, Salsa-20, Serpent, Twofish keys | R |
| Digital Signature Generation and Verification | Sign or verify operations with RSA, DSA, ECDSA key sizes and elliptic curves not listed in Table 7 or Table 8; signature generation with SHA-1. | ✓ | | RSA, DSA, ECDSA key pairs | C, R, U |

| Service | Service Description and Algorithms | Role | | Keys | Access Types |
|---|---|---|---|---|---|
| | | U | CO | | |
| TLS Key Agreement | Negotiate a TLS key agreement to establish a secure channel with key sizes or curves not listed in Table 7 or Table 8. | ✓ | | RSA, Diffie-Hellman, EC Diffie-Hellman key pairs, pre-master secret, master secret, derived keys (AES, Triple-DES, HMAC, and other algorithms in Table 9), KDF internal state | C, R, U |
| TLS Network Protocol v1.0, v1.1, v1.2 | Provide data encryption and authentication over TLS network protocol.<br>Ciphersuites composed of any algorithms *not* listed in Table 7 or Table 8. | ✓ | | AES key<br>Triple-DES key<br>HMAC key<br>Pre-master secret<br>Master secret<br>Diffie-Hellman public/private key pair<br>EC Diffie-Hellman public/private key pair<br>RSA, DSA, ECDSA public/private keys | C, R, U |
| Diffie-Hellman Key Agreement | Establish a shared secret with keys not listed in Table 7 or Table 8. | ✓ | | Diffie-Hellman key pair | C, R, U |
| EC Diffie-Hellman Key Agreement | Establish a shared secret with curves not listed in Table 7 or Table 8. | ✓ | | EC Diffie-Hellman key pair | C, R, U |
| Asymmetric Key Generation | Generation RSA, DSA, ECDSA keys in sizes no listed in Table 7 and not compliant with FIPS 186-4. | ✓ | | RSA, DSA, ECDSA key pairs | C, R, U |
| Random Number Generation | Generation of random numbers using the lagged Fibonacci PRNG, Yarrow RNG, and ANSI X9.31 RNG. | ✓ | | Entropy input string and internal state | C, R, U |
| Message Digest | Hashing using non-Approved hash functions (e.g., GOST, MD2, MD4, MD5, MDC2, RIPEMD160, SHA3, Whirlpool). | ✓ | | n/a | n/a |
| Message Authentication Code | MAC generation using UMAC, GMAC or HMAC using keys not listed in Table 7. | ✓ | | MAC key | R |
| Password-Based Key Derivation Function | Key derivation from password using PBKDF2 | ✓ | | Derived keys | C, R, U |

| Service | Service Description and Algorithms | Role | | Keys | Access Types |
|---|---|---|---|---|---|
| | | U | CO | | |
| DANE Certificate Management and Protocol | DNS-based Authentication of Named Entities (DANE) protocol for binding X.509 certificates to DNS names using DNSSEC. | ✓ | | RSA, DSA, ECDSA key pairs. | C, R, U |
| OpenPGP Certificate Management | OpenPGP certificates within TLS | ✓ | | RSA, DSA, ECDSA key pairs | C, R, U |
| PKCS#11 Certificate Management | PKCS#11 certificates in GnuTLS | ✓ | | RSA, DSA, ECDSA key pairs | C, R, U |
| SRTP Protocol Support | Support for SRTP (RFC 5764) | ✓ | | AES, HMAC keys | C, R, U |
| Trusted Platform Module Support | Support for TPM 1.2 | ✓ | | RSA, DSA, ECDSA key pairs | C, R, U |

## 4.3 Algorithms

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

The module supports different AES and SHS implementations based on the underlying platform's capability (per the tested operational environment in Table 3). The module supports the use of AES-NI and SSSE3 from the Intel architecture. When the AES-NI is enabled in the operating environment, the module performs the AES operations supported by the AES-NI instructions. When the AES-NI is disabled in the operating environment, the module performs the AES operations using the support from the Supplemental Streaming SIMD Extensions 3 (SSSE3). The module also performs SHS operations using the support from the SSSE3.

The AES and SHS implementations that use the AES-NI and SSSE3 instructions and their related algorithms have been tested by CAVS and subjected to functional testing. Although the module offers different implementations for AES and SHS, only one implementation for the respective algorithm will ever be available for AES, SHS and HMAC cryptographic services at run-time.

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

### 4.3.1 FIPS-Approved Algorithms

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

*Table 7: FIPS-approved cryptographic algorithms.*

| Algorithm | Standard | Mode/Method | Key size | Use | CAVP Cert# |
|---|---|---|---|---|---|
| AES | [FIPS197] [SP800-38A] | CBC | 128, 192 and 256 bits | Data Encryption and Decryption | #C790 (SSSE3) #C791 (AES-NI) #C792 (C) |
| | [FIPS197] [SP800-38D] | GCM | 128 and 256 bits | Data Encryption and Decryption | |
| | [FIPS197] [SP800-38D] | GMAC | 128 and 256 bits | Message Authentication Code | |
| DSA | [FIPS 186-4] | | L=2048, N=224; L=2048, N=256; L=3072, N=256 | Key Pair Generation | #C792 (C) |
| | | SHA2-384 P/Q Generation: Provable G Generation: Canonical | L=2048, N=224; L=2048, N=256; L=3072, N=256 | Domain Parameter Generation Domain Parameter Verification | |
| | | SHA2-224, SHA2-256, SHA2-384, SHA2-512 | L=2048, N=224 | Signature Generation | |
| | | SHA2-256, SHA2-384, SHA2-512 | L=2048, N=256; L=3072, N=256 | | |
| | | SHA2-224, SHA2-256, SHA2-384, SHA2-512 | L=1024, N=160, L=2048, N=224 | Signature Verification | |
| | | SHA2-256, SHA2-384, SHA2-512 | L=2048, N=256; L=3072, N=256 | | |
| DRBG | [SP800-90A] | CTR_DRBG AES-256 without DF, without PR | n/a | Random Number Generation | #C792 (C) Prerequisite AES #C789 (Nettle) |
| ECDSA | [FIPS186-4] | Testing Candidates | P-256, P-384, P-521 | Key Pair Generation | #C792 (C) |
| | | | P-256, P-384, P-521 | Public Key Verification | |

| Algorithm | Standard | Mode/Method | Key size | Use | CAVP Cert# |
|---|---|---|---|---|---|
| | | SHA2-224, SHA2-256, SHA2-384, SHA2-512 | P-256, P-384, P-521 | Signature Generation | |
| | | SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512 | P-256, P-384, P-521 | Signature Verification | |
| KAS ECC Component (CVL) | [SP800-56A] | ECC Ephemeral Unified scheme | P-256 (EC), P-384 (ED), P-521 (EE) | EC Diffie-Hellman Key Agreement | #C792 (C) |
| KAS FFC Component (CVL) | [SP800-56A] | FFC dhEphem scheme | p=2048, q=224 (FB);<br>p=2048, q=256 (FC) | Diffie-Hellman Key Agreement | #C792 (C) |
| HMAC | [FIPS198-1] | SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512 | 112 bits or greater | Message Authentication Code | #C792 (C) |
| KDF Component in TLS v1.0/1.1 TLS v1.2 (CVL) | [SP800-135] | SHA2-256, SHA2-384, SHA2-512 | | Key Derivation | #C792 (C) |
| RSA | [FIPS186-4] | B.3.2 Provaby Primes | 2048 and 3072 bits | Key Pair Generation | #C792 (C) |
| | | PKCS#1v1.5 with SHA2-224, SHA2-256, SHA2-384, SHA2-512 | 2048 and 3072 bits | Digital Signature Generation | |
| | | PKCS#1v1.5 with SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512 | 1024, 2048, and 3072 bits | Signature Verification | |
| SHS | [FIPS180-4] | SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512 | N/A | Message Digest | #C792 (C)<br>#C790 (SSSE3) |

| Algorithm | Standard | Mode/Method | Key size | Use | CAVP Cert# |
|---|---|---|---|---|---|
| Triple-DES | [SP800-67] [SP800-38A] | CBC | 192 bits | Data Encryption and Decryption | #C792 (C) |
| KTS | [SP800-38F] | AES-GCM | 128, 256 bits | Key Wrapping | #C790 (SSSE3) #C791 (AES-NI) #C792 (C) |
| | [SP800-38F] [FIPS198-1] | AES-CBC and HMAC | 128, 192, 256 bits | Key Wrapping | #C790 (SSSE3) #C791 (AES-NI) #C792 (C) |
| | [SP800-38F] [FIPS198-1] | Triple-DES-CBC and HMAC | 192 bits | Key Wrapping | #C792 (C) |
| **Nettle Library** | | | | | |
| AES[2] | [FIPS197] | ECB | 128, 192 and 256 bits | Data Encryption and Decryption | #C789 |

### 4.3.2 Non-Approved-but-Allowed Algorithms

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

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

| Algorithm | Usage |
|---|---|
| RSA Key Wrapping with key size between 2048 bits and 15360 bits (or more) | Key wrapping, key establishment methodology provides between 112 and 256 bits of encryption strength. |
| Diffie-Hellman with key size between 2048 bits and 15360 bits (or more) | Key agreement, key establishment methodology provides between 112 and 256 bits of encryption strength. |
| EC Diffie-Hellman with P-256, P-384, P-521 curves | Key agreement, key establishment methodology provides between 128 and 256 bits of encryption strength. |
| NDRNG | Used for seeding NIST SP 800-90A DRBG. |
| MD5 | Message digest used only in TLS. |

[2] This implementation is used internally by the module, but the algorithm is not directly accessible to operators of the module.

### 4.3.3 Non-Approved Algorithms

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

*Table 9: Non-FIPS approved cryptographic algorithms.*

| Algorithm | Usage |
|---|---|
| ANSI X9.31 RNG | Random Number Generation |
| Blowfish | Encryption/decryption |
| Camellia | Encryption/decryption |
| CAST128 | Encryption/decryption |
| DES | Encryption/decryption |
| Diffie-Hellman | Key agreement using keys of length not listed in Table 7 |
| DSA | Parameter/Key generation/Signature generation with keys not listed in Table 7 |
| EC Diffie-Hellman | Key agreement using curves not listed in Table 7 |
| ECDSA | Key generation/Signature generation with curves not listed in Table 7 |
| GOST | GOST Hash R 34.11-94 (RFC4357) |
| Lagged Fibonacci Pseudo-randomness Generator | Generating random numbers |
| MD2 | Hash function |
| MD4 | Hash function |
| MD5 | Hash function |
| MDC2 | Hash function |
| PBKDF2 | Password-based key derivation |
| RC2 | Encryption/decryption |
| RC4 | Encryption/decryption |
| RIPEMD160 | Hash function |
| RSA | Key generation/Signature generation/Signature verification with keys of length not listed in Table 7 |
| Salsa20 | Encryption/decryption |
| Serpent | Encryption/decryption |
| SHA-1 | Signature generation |

| Algorithm | Usage |
|---|---|
| SHA3 | Hash function |
| Twofish | Encryption/decryption |
| UMAC | Message authentication code |
| Whirlpool | Hash function |
| Yarrow RNG | Random number generation |

## 4.4 Operator Authentication

The module does not support operator authentication mechanisms. The role of the operator is implicitly assumed based on the service requested.

# 5 Physical Security

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

# 6 Operational Environment

## 6.1 Applicability

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

## 6.2 Policy

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

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

In operational mode, the ptrace(2) system call, the 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 Cryptographic Key Management

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

All key and CSP storage are done in plaintext.

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

| Name | Use | Generation/ Establishment | Entry/ Output | Type | Stored In | Zeroization |
|---|---|---|---|---|---|---|
| AES Key | Encryption, decryption.<br>MAC generation and verification (GMAC). | Provided by the calling application. | Entered via API input parameter.<br>No output. | AES key, all modes per Table 7.<br>Length: 128, 192, 256 bits for CBC; 128, 256 bits for GCM, GMAC. | RAM | gnutls_cipher_deinit() |
| Triple-DES Keys | Encryption, decryption. | Provided by the calling application. | Entered via API input parameter.<br>No output. | CBC, 192 bits. | RAM | gnutls_cipher_deinit() |
| HMAC Key | MAC generation and verification | Provided by the calling application. | Entered via API input parameter.<br>No output. | HMAC keys of length > 112 bits. | RAM | gnutls_hmac_deinit() |
| RSA public and private key | RSA signature generation and verification.<br>Key wrapping. | Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800-90A DRBG. | Entered via API input parameter or generated by module.<br>Output via API output parameters in plaintext. | RSA keys of length 1024, 2048, 3072 bits (or more as allowed for key wrapping) | RAM | gnutls_rsa_params_deinit()<br>gnutls_privkey_deinit()<br>gnutls_x509 _privkey_deinit()<br>gnutls_pubkey_deninit() |
| DSA public and private key | DSA signature generation and verification. | Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800-90A DRBG. | Entered via API input parameter or generated by module.<br>Output via API output parameters in plaintext. | DSA keys of length 1024, 2048, 3072 bits | RAM | gnutls_privkey_deinit()<br>gnutls_pubkey_deninit()<br>gnutls_x509_privkey_deinit()<br>gnutls_x509_crt_deinit() |
| ECDSA public and private key | ECDSA signature generation and verification. | Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800-90A DRBG. | Entered via API input parameter or generated by module.<br>Output via API output parameters in plaintext. | ECDSA keys for curves P-256, P-384, P-521 | RAM | gnutls_privkey_deinit()<br>gnutls_pubkey_deninit()<br>gnutls_x509_privkey_deinit() |
| Entropy input string | Entropy input strings used to construct the seed for the DRBG. | Obtained from NDRNG. | N/A | 384 bits | RAM | gnutls_global_deinit() |

| Name | Use | Generation/ Establishment | Entry/ Output | Type | Stored In | Zeroization |
|---|---|---|---|---|---|---|
| DRBG Internal state (V, Key) | Used internally by CTR DRBG. Used to generate random bits. | During DRBG initialization. | N/A | | RAM | gnutls_global_deinit() |
| **TLS Network Protocol** | | | | | | |
| TLS KDF internal state | Values of the TLS KDF internal state used in TLS tunnel establishment | SP800-135 TLS KDF | N/A | | RAM | gnutls_deinit() |
| AES Derived Key | Encryption, decryption.<br>MAC generation and verification (GMAC). | Generated internally by the module (from the [SP800-135] TLS KDF) during the establishment of the TLS tunnel. | N/A | AES key (CBC, GCM) with length 128 or 256 bits (defined by ciphersuite). | RAM | gnutls_cipher_deinit() |
| Triple-DES Derived Keys | Encryption, decryption. | Generated internally by the module (from the [SP800-135] TLS KDF) during the establishment of the TLS tunnel. | N/A | CBC, 192 bits. | RAM | gnutls_cipher_deinit() |
| HMAC Derived Key | MAC generation and verification | Generated internally by the module (from the [SP800-135] TLS KDF) during the establishment of the TLS tunnel. | N/A | HMAC key with length defined by ciphersuite. | RAM | gnutls_hmac_deinit() |
| Pre-master secret | Establishment of encrypted session as input to the derivation of the master secret. | Generated during the key agreement when using Diffie-Hellman or EC Diffie-Hellman key exchange.<br>Generated by TLS client as output from DRBG when using RSA key exchange. | Entry: if received by module as TLS server, wrapped with server's public RSA key; otherwise no entry.<br>Output: if generated by module as TLS client, wrapped with server's public RSA key; otherwise, no output. | Length defined per ciphersuite | RAM | gnutls_deinit() |
| Master secret | Establishment of encrypted session. | Derived from pre-master secret. | Generated by the module. No output. | 384 bits | RAM | gnutls_deinit() |
| Diffie-Hellman public and | Key agreement. | Keys are generated using FIPS 186-4 and the random value used in the key generation is | Entered via API parameters. Output via API | Key lengths 2048 bits and 15360 bits (or more) | RAM | gnutls_deinit()<br>gnutls_dh_params_deinit() |

| Name | Use | Generation/ Establishment | Entry/ Output | Type | Stored In | Zeroization |
|---|---|---|---|---|---|---|
| private key | | obtained from SP800-90A DRBG. | parameters in plaintext. | | | |
| EC Diffie-Hellman public and private key | Key agreement. | Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800-90A DRBG. | Entered via API parameters. Output via API parameters in plaintext. | Curves P-256, P-384, P-521 | RAM | gnutls_deinit() gnutls_dh_params_deinit() |

## 7.1 Random Number Generation

The module provides a DRBG compliant with [SP800-90A] for the creation of key components of asymmetric keys, and random number generation. The DRBG implements a CTR_DRBG mechanism with AES-256 without derivation function and without prediction resistance. The CTR_DRBG is implemented in the libgnutls library and provides at least 128 bits of output data per each request.

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

The module performs continuous random number generator tests (CRNGT) on the output of SP800-90A DRBG to ensure that consecutive random numbers do not repeat. Moreover, the module performs the health tests for the SP800-90A DRBG as defined per Section 11.3 of SP800-90A.

The operational environment, the Linux RNG, performs the continuous test on the NDRNG.

## 7.2 Key Generation

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

Symmetric keys are derived from the shared secret established by Diffie-Hellman and EC Diffie-Hellman in a manner that is compliant to NIST SP 800-135 for TLS KDF.

## 7.3 Key Entry and Output

The module does not support manual key entry or intermediate key generation output. In addition, the module does not produce key output outside its physical boundary. The keys can be entered or output from the module in plaintext form via API parameters, to and from the calling application only. The module provides services to import and export public and private keys to and from calling applications only.

## 7.4 Key/CSP Storage

Public and private keys are provided to the module by the calling process, and are destroyed when released by the appropriate API function calls. The module does not perform persistent storage of keys. The keys and CSPs are stored as plaintext in volatile memory (RAM). The protection of these keys and CSPs in RAM is provided by the operating system enforcement of separation of address space.

The HMAC keys used for integrity tests are stored within the module's binary files and rely on the operating system for protection.

## 7.5 Key/CSP Zeroization

The application is responsible for calling the appropriate destruction functions from the API to zeroize keys and CSPs. The destruction functions then overwrite the memory occupied by keys with zeros and deallocates the memory with the free() call. In case of abnormal termination, the keys in physical memory are overwritten by the Linux kernel before the physical memory is allocated to another process.

## 7.6 Key Establishment

The module provides Diffie-Hellman and EC Diffie-Hellman key agreement schemes from [SP800-56A]. These key agreement schemes are also used as part of the TLS protocol key exchange. The module also provides RSA key wrapping (encapsulation) using public key encryption and private key decryption primitives as allowed by [FIPS140-2_IG] D.9. RSA key wrapping may be used as part of the TLS protocol key exchange.

The module provides approved key transport methods according to IG D.9, which can be used in the TLS protocol context. The key transport methods are provided either by using an approved authenticated encryption mode (e.g., AES-GCM) or a combination method. The combination method consists of using an approved symmetric encryption mode (e.g., AES-CBC or Triple-DES CBC) together with an approved authentication method (e.g., HMAC).

Table 7 and Table 8 specify the key sizes allowed in the FIPS mode of operation. According to “Table 2: Comparable strengths” in [SP800-57], the key sizes of key wrapping and transport, RSA, Diffie-Hellman and EC Diffie-Hellman provide the following security strengths:

- RSA key wrapping provides between 112 and 256 bits of encryption strength.
- Diffie-Hellman key establishment methodology provides between 112 and 256 bits of encryption strength.
- EC Diffie-Hellman key establishment methodology provides between 128 and 256 bits of encryption strength.
- Approved authenticated encryption mode (i.e. GCM) key establishment methodology provides 128 or 256 bits of encryption strength.
- Combination of approved AES encryption and HMAC authentication key establishment methodology provides between 128 and 256 bits of encryption strength.
- Combination of approved Triple-DES encryption and HMAC authentication key establishment methodology provides 112 bits of encryption strength.

## 7.7 Handling of Keys and CSPs between Modes of Operation

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

# 8 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

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

# 9 Self-Tests

## 9.1 Power-on Self-Tests

The module performs power-up or power-on self-tests (POSTs) automatically during loading of the module by making use of default entry point (DEP). These POSTs ensure that the module is not corrupted and that the cryptographic algorithms work as expected. No operator intervention is necessary to run the POSTs. The self-tests cover different implementations depending on the availability of those implementations in the operational environment (e.g., AES-NI, SSSE3).

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

The integrity of the module binary is verified using an HMAC-SHA2-256. The HMAC value is computed at build time and stored in the .hmac file. The value is recalculated at runtime and compared against the stored value in the file. If the comparison succeeds, then the remaining POSTs (consisting of the algorithm-specific Known Answer Tests) are performed. On successful completion of the all the power-on tests, the module becomes operational and crypto services are then available. If any of the tests fails, the module transitions to the error state and subsequent calls to the module will fail. Thus, in the error state, no further cryptographic operations will be possible.

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

*Table 11: Self-tests.*

| Algorithm | Test |
|---|---|
| AES | • KAT AES-CBC with 128-bit key, encryption<br>• KAT AES-CBC with 128-bit key, decryption<br>• KAT AES-GCM with 256-bit key, encryption<br>• KAT AES-GCM with 256-bit key, decryption |
| Triple-DES | • KAT Triple-DES (CBC) with 192-bit key, encryption<br>• KAT Triple-DES (CBC) with 192-bit key, decryption |
| DSA | • KAT DSA with 2048-bit key and SHA2-256 |
| RSA | • KAT RSA PKCS#1v1.5 signature generation and verification with 2048-bit key and using SHA2-256 |
| ECDSA | • KAT ECDSA with P-256 and SHA2-256 |
| KAS FFC (Diffie-Hellman) | • Primitive "Z" Computation KAT with 2048-bit key |
| KAS ECC (EC Diffie-Hellman) | • Primitive "Z" Computation KAT with P-256 curve |
| DRBG | • KAT CTR_DRBG using AES-256 without DF, without PR (this self-test also covers the self-test for AES-ECB in the Nettle implementation) |
| HMAC | • KAT HMAC-SHA-1<br>• KAT HMAC-SHA2-224<br>• KAT HMAC-SHA2-256 |

| Algorithm | Test |
|---|---|
| | • KAT HMAC-SHA2-384<br>• KAT HMAC-SHA2-512 |
| SHS | • Performed as part of the HMAC KAT, allowed by IG 9.1 and IG 9.4 [FIPS140-2_IG] |
| Module Integrity | • HMAC-SHA2-256 |

## 9.2 Conditional Self-Tests

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

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

*Table 12: Conditional self-tests.*

| Algorithm | Test |
|---|---|
| DSA Key generation | PCT, signature generation and verification |
| ECDSA Key generation | PCT, signature generation and verification |
| RSA Key generation | PCT, signature generation and verification, and for encryption and decryption |
| DRBG | Continuous Random Number Generator Test (CRNGT) |

## 9.3 On-Demand self-tests

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

# 10 Guidance

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

## 10.1 Crypto-Officer Guidance

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

For proper operation of the in-module integrity verification, the prelink has to be disabled. This can be done by setting PRELINKING=no in the /etc/sysconfig/prelink configuration file. If the module is already prelinked, the prelink should be undone on all the system files using the 'prelink -u -a' command.

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

1. Install the dracut-fips package:

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

2. Recreate the INITRAMFS image:

   ```
   # dracut –f
   ```

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

```
fips=1
```

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

```
"df /boot"
```

or

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

For example:

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

The partition of /boot is located on /dev/sda1 in this example. Therefore, the following string needs to be appended to the kernel command line:

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

Reboot to apply these settings.

The Crypto Officer shall check whether the file /proc/sys/crypto/fips_enabled exists and whether it contains “1”. If the file does not exist or does not contain “1”, the operational environment is not configured to support FIPS and the module will not operate as a FIPS validated module. Once the operational environment has been configured to support FIPS, it is not possible to switch back to standard mode without terminating the module first.

After performing the above configuration, the Crypto Officer should proceed to module installation. The RPM package of the module can be installed using standard tools recommended for the installation of packages on an Amazon Linux 2 system (e.g., yum, RPM). The integrity of the RPM is automatically verified during the installation of the module and the Crypto Officer shall not install the RPM file if the yum server indicates an integrity error.

## 10.2 User Guidance

The applications must be linked dynamically to run the module. Only the services listed in Table 5 are allowed to be used in FIPS mode.

The libraries of GMP and Nettle provide the support of cryptographic operations to the GnuTLS library. The operator shall use the API provided by the GnuTLS library for the services. Directly invoking the APIs provided by the supporting libraries is forbidden and implicitly switches the module to the non-FIPS mode.

### 10.2.1 TLS and Diffie-Hellman

The TLS protocol implementation provides both the server and the client sides. As required by SP800-131A, 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 cryptographic module accepts Diffie-Hellman key sizes smaller than 2048 bits offered by the TLS server.

To comply 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 TLS server, the Diffie-Hellman domain parameters must be of 2048 bits or larger.
- In case the module is used as TLS client, the TLS server must be configured to only offer Diffie-Hellman domain parameters of 2048 bits or larger.

### 10.2.2 AES GCM IV

AES GCM encryption and decryption are used in the context of the TLS protocol version 1.2 (compliant to Scenario 1 in [FIPS140-2_IG] A.5). The module is compliant with [SP 800-52] and the mechanism for IV generation is compliant with [RFC5288]. The operations of one of the two parties involved in the TLS key establishment scheme are performed entirely within the cryptographic boundary of the module, including the setting of the counter portion of the IV.

When the nonce_explicit part of the IV exhausts the maximum number of possible values for a given session key, the module (acting as server or client) triggers a handshake to establish a new encryption key per Section 7.4.1.1 and Section 7.4.1.2 in [RFC5246] and compliant to [FIPS140-2_IG] A.5.

In case the module's power is lost and then restored, the key used for AES GCM encryption or decryption shall be re-distributed.

### 10.2.3 Triple-DES Data Encryption

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

### 10.2.4 Key Usage and Management

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

- The DRBG engine shall be reseeded.

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

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

## 10.3 Handling Self-Test Errors

The module transition to error state when any of self-tests or conditional tests fails. The module then returns an error code to indicate the error, and further cryptographic operations are inhibited.

Table 13 lists the error events, error codes and error messages respective to failures during self-tests and when the module is in the error state.

*Table 13: Error events, codes and messages.*

| Error Event | Error Code | Error Message |
|---|---|---|
| When the KAT or Integrity fails at the power-on | GNUTLS_E_SELF_TEST_ERROR (-400) | “Error while performing self checks.” |
| When the KAT of DRBG fails at the power-on | 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.” |

Self-test errors transition the module into an error state that keeps the module active, but prevents any cryptographic related operations. The module must be restarted and perform power-on 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 https://gnutls.org/manual/html_node/Error-codes.html.

# 11 Mitigation of Other Attacks

RSA is vulnerable to timing attacks [Brumley; Boneh, 2003]. In a setup in which attackers can measure the time of RSA decryption or signature operations, blinding must be used to protect the RSA operation from that attack. The API function _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 includes this random factor in those RSA operations such that timing information is made unusable, mitigating the RSA timing attacks.

# 12 Acronyms, Terms and Abbreviations

| Term | Definition |
|---|---|
| AES | Advanced Encryption Standard |
| AESNI | Advanced Encryption Standard New Instructions |
| CAVP | Cryptographic Algorithm Validation Program |
| CMVP | Cryptographic Module Validation Program |
| CSE | Communications Security Establishment |
| CSP | Critical Security Parameter |
| DANE | DNS-based Authentication of Named Entities |
| DH | Diffie-Hellman |
| DHE | Diffie-Hellman Ephemeral |
| DRBG | Deterministic Random Bit Generator |
| DTLS | Datagram Transport Layer Security |
| ECDH | Elliptic Curve Diffie-Hellman |
| ECDSA | Elliptic Curve Digital Signature Algorithm |
| EDC | Error Detection Code |
| HMAC | (Keyed) Hash Message Authentication Code |
| IKE | Internet Key Exchange |
| KAT | Known Answer Test |
| KDF | Key Derivation Function |
| NDRNG | Non-Deterministic Random Number generator |
| NIST | National Institute of Standards and Technology |
| PAA | Processor Algorithm Acceleration |
| POST | Power On Self-Test |
| PR | Prediction Resistance |
| PSS | Probabilistic Signature Scheme |
| PUB | Publication |
| SHA2 | Secure Hash Algorithm |
| SSSE3 | Supplemental Streaming SIMD Extensions 3 |
| TLS | Transport Layer Security |

# 13 References


**Brumley; Boneh** **Brumley, David; Boneh, Dan. Remote Timing Attacks are Practical.**
SSYM'03 Proceedings of the 12th Conference on USENIX Security Symposium, 2003.
https://dl.acm.org/citation.cfm?id=1251354

**FIPS140-2** **FIPS PUB 140-2 - Security Requirements Forfor Cryptographic Modules**
May 2001
http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

**FIPS140-2_IG** **Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program**
May 7, 2019
https://csrc.nist.gov/CSRC/media/Projects/cryptographic-module-validation-program/documents/fips140-2/FIPS1402IG.pdf

**FIPS180-4** **Secure Hash Standard (SHS)**
March 2012
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

**FIPS186-4** **Digital Signature Standard (DSS**
July 2013
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

**FIPS197** **Advanced Encryption Standard**
November 2001
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

**FIPS198-1** **The Keyed Hash Message Authentication Code (HMAC)**
July 2008
http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

**PKCS#1** **Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.2**
November 2016
https://tools.ietf.org/rfc/rfc8017.txt

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

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

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

| | |
|---|---|
| **RFC5246** | **The Transport Layer Security (TLS) Protocol Version 1.2**<br>August 2008<br>https://tools.ietf.org/html/rfc5246 |
| **RFC5288** | **AES Galois Counter Mode (GCM) Cipher Suites for TLS**<br>August 2008<br>https://tools.ietf.org/html/rfc5288 |
| **RFC5764** | **Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)**<br>May 2010<br>https://tools.ietf.org/html/rfc5764 |
| **SP800-131A** | **NIST Special Publication 800-131A Revision 2- Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths**<br>March 2019<br>https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf |
| **SP800-135** | **NIST Special Publication 800-135 Revision 1 - Recommendation for Existing Application-Specific Key Derivation Functions**<br>December 2011<br>http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf |
| **SP800-38A** | **NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques**<br>December 2001<br>http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf |
| **SP800-52** | **NIST Special Publication 800-52 Revision 1 - Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations**<br>April 2014<br>http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf |
| **SP800-56A** | **NIST Special Publication 800-56A - Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)**<br>March, 2007<br>http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf |
| **SP800-67** | **NIST Special Publication 800-67 Revision 2 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher**<br>November 2017<br>https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf |
| **SP800-90A** | **NIST Special Publication 800-90A - Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators**<br>June 2015<br>http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf |