SUSE Linux Enterprise Server GnuTLS Cryptographic Module

version 1.0

FIPS 140-2 Non-Proprietary Security Policy

Doc version 1.0.4

Last update: 2021-11-23

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of contents

1 Cryptographic Module Specification

This document is the non-proprietary security policy for the SUSE Linux Enterprise Server GnuTLS Cryptographic Module version 1.0. It contains the security rules under which the module must operate 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.

This document was prepared in partial fulfillment of the FIPS 140-2 requirements for cryptographic modules and is intended for security officers, developers, system administrators and end-users.

FIPS 140-2 details the requirements of the Governments of the U.S. and Canada for cryptographic modules, aimed at the objective of protecting sensitive but unclassified information. For more information on the FIPS 140-2 standard and validation program please refer to the NIST website at http://csrc.nist.gov/.

Throughout the document, “the GnuTLS module” and “the module” are also used to refer to the SUSE Linux Enterprise Server GnuTLS Cryptographic Module version 1.0.

1.1 Module Overview

The SUSE Linux Enterprise Server GnuTLS Cryptographic Module is a set of libraries implementing general purpose cryptographic algorithms and network protocols. The module supports the Transport Layer Security (TLS) protocol defined in [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.

For the purpose of the FIPS 140-2 validation, the module is a software-only, multi-chip standalone cryptographic module validated at overall security level 1. Table 1 shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2 standard:

FIPS 140-2 Section		Security Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services and Authentication	1
4	Finite State 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	N/A

Table 1: Security Levels

Table 2 lists the software components of the cryptographic module, which defines its logical boundary.

Component	Description
/usr/lib64/libgnutls.so.30	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.6	Provides the cryptographic algorithm implementations, including AES, Triple-DES, SHA, HMAC, RSA Digital Signature, DSA and ECDSA.
/usr/lib64/libhogweed.so.4	Provides primitives used by libgnutls and libnettle to support the asymmetric cryptographic operations.
/usr/lib64/libgmp.so.10	Provides big number arithmetic operations to support the asymmetric cryptographic operations.
/usr/lib64/.libgnutls.so.30.hmac	The .hmac files contain the HMAC-SHA-256 values of their associated library for integrity check during the power-up.
/usr/lib64/.libnettle.so.6.hmac	
/usr/lib64/.libhogweed.so.4.hmac	
/usr/lib64/.libgmp.so.10.hmac	

Table 2: Cryptographic Module Components

The block diagrams below shows the module's logical boundary, its interface with the operational environment and the delimitation of its logical boundary.

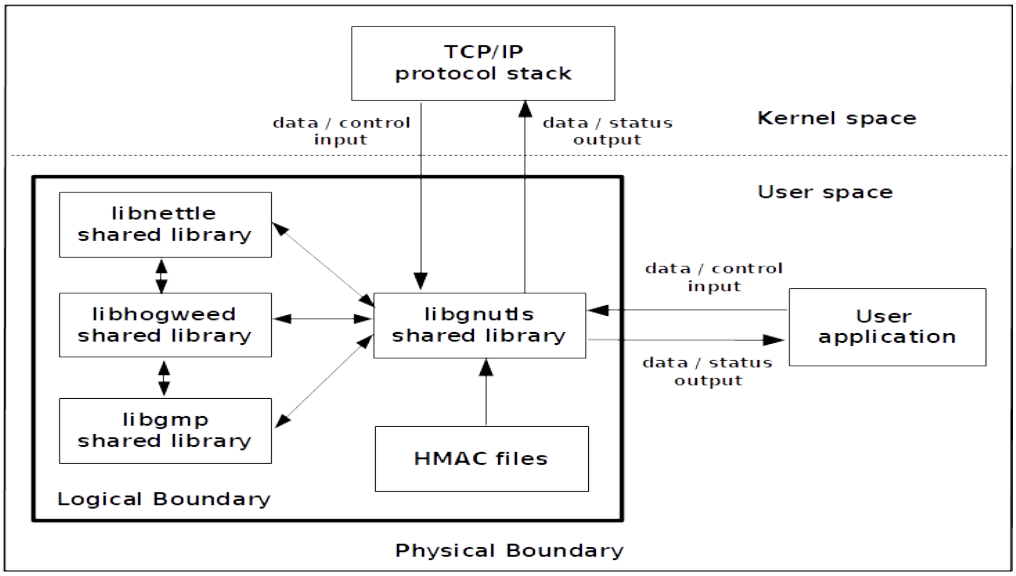

Figure 1: Software Block Diagram

The module is aimed to run on a general purpose computer (GPC). Table 3 shows the platform on which the module has been tested:

Platform	Processor	Test Configuration
Dell EMC PowerEdge 640	Intel Cascade Lake Xeon Gold 6234	SUSE Linux Enterprise Server 15 SP2 with and without AES-NI (PAA)
IBM System Z/15	IBM z15	SUSE Linux Enterprise Server 15 SP2
Gigabyte R181-T90	Cavium ThunderX2 CN9975 ARMv8	SUSE Linux Enterprise Server 15 SP2 with and without Crypto Extensions (PAA)

Table 3: Tested Platforms

Note: Per FIPS 140-2 IG G.5, the Cryptographic Module Validation Program (CMVP) makes no statement as to the correct operation of the module or the security strengths of the generated keys when this module is ported and executed in an operational environment not listed on the validation certificate.

The physical boundary of the module is the surface of the case of the tested platform. Figure 2 shows the hardware block diagram including major hardware components of a GPC.

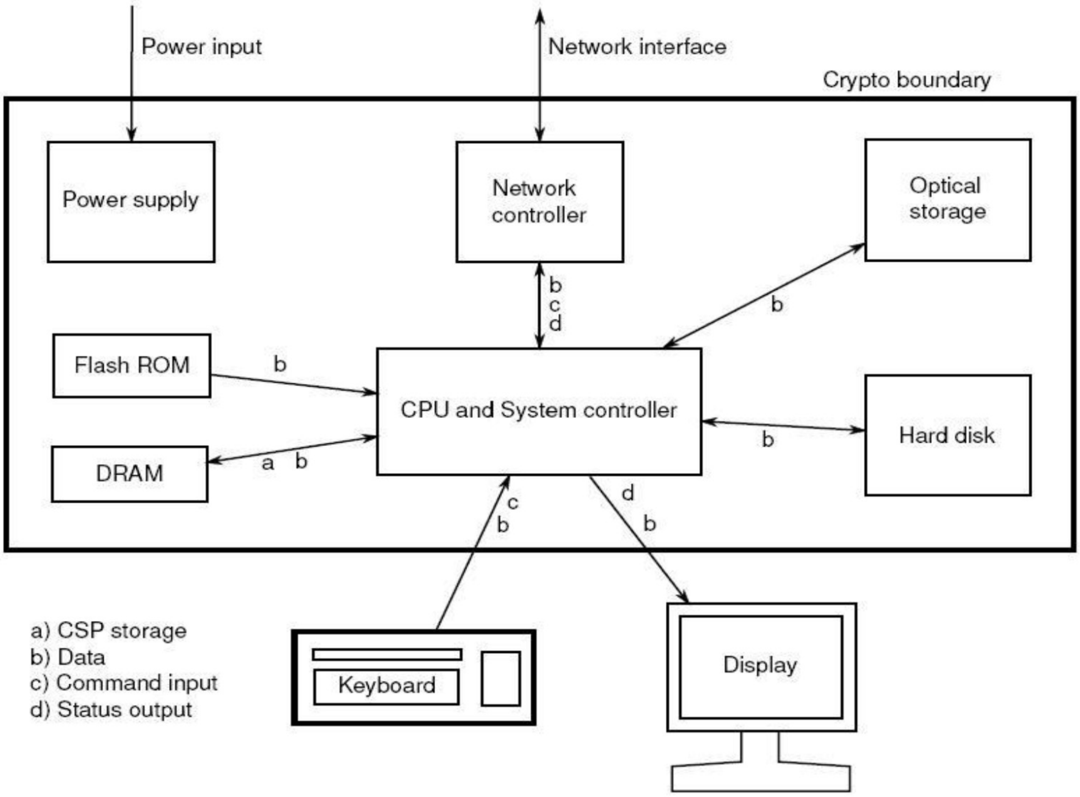

Figure 2: Hardware Block Diagram

1.2 Modes of Operation

The module supports two modes of operation:

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

The module enters FIPS mode after power-up tests succeed. Once the module is operational, the mode of operation is implicitly assumed depending on the security function invoked and the security strength of the cryptographic keys.

Critical security parameters (CSPs) used or stored in FIPS mode are not used in non-FIPS mode, and vice versa.

2 Cryptographic Module Ports and Interfaces

As a software-only module, the module does not have physical ports. For the purpose of the FIPS 140-2 validation, the physical ports are interpreted to be the physical ports of the hardware platform on which it runs.

The logical interfaces are the API through which applications request services, and the TLS protocol internal state and messages sent and received from the TCP/IP protocol. The ports and interfaces are shown in the following table.

FIPS Interface	Physical Port	Logical Interface
Data Input	Ethernet ports	API input parameters, kernel I/O network or files on filesystem, TLS protocol input messages.
Data Output	Ethernet ports	API output parameters, kernel I/O network or files on filesystem, TLS protocol output messages.
Control Input	Ethernet port	API function calls, API input parameters for control.
Status Output	Ethernet port	API return values.
Power Input	PC Power Supply Port	N/A

Table 4: Ports and Interfaces

3 Roles, Services and Authentication

3.1 Roles

The module supports the following roles:

- User role: performs cryptographic services (in both FIPS mode and non-FIPS mode), TLS network protocol, key zeroization, get status, and on-demand self-test.
- Crypto Officer role: performs module installation and configuration, certificate management.

3.2 Services

The module provides services to the users that assume one of the available roles. All services are shown in Table 5 and Table 6.

Table 5 lists the services available in FIPS mode. For each service, the table lists the associated cryptographic algorithm(s), the role to perform the service, the cryptographic keys or CSPs involved, and their access type(s). The following convention is used to specify access rights to a CSP:

- *Create*: the calling application can create a new CSP.
- *Read*: the calling application can read the CSP.
- *Update*: the calling application can write a new value to the CSP.
- *Zeroize*: the calling application can zeroize the CSP.
- *n/a*: the calling application does not access any CSP or key during its operation.

The details of the approved cryptographic algorithms including the CAVP certificate numbers can be found in Table 7.

Service	Algorithm	Role	Keys/CSPs	Access
Cryptographic Services				
Symmetric key generation	DRBG	User	AES, Triple-DES and HMAC keys	Create
Symmetric encryption and decryption	AES	User	AES key	Read
	Three-key Triple-DES	User	Three-key Triple-DES key	Read
Symmetric decryption	Two-key Triple-DES	User	Two-key Triple-DES key	Read
Asymmetric key generation in X.509 certificate	RSA, DSA, ECDSA	User	RSA public and private keys	Create
			DSA public and private keys	
			ECDSA public and private keys	
Digital signature generation in X.509 certificate	RSA, DSA, ECDSA, SHS	User	RSA private key	Read
			DSA private key	
			ECDSA private key	
Digital signature verification in X.509 certificate	RSA, DSA, ECDSA, SHS	User	RSA public key	Read
			DSA public key	
			ECDSA public key	

Service	Algorithm	Role	Keys/CSPs	Access
DSA Domain Parameter Generation	DSA	User	None	n/a
Public key verification	DSA, ECDSA, RSA	User	DSA, ECDSA or RSA public key	Read
Import public key	N/A	User	DSA, ECDSA or RSA public key	Create
Export public key	N/A	User	DSA, ECDSA or RSA public key	Read
Import private key	N/A	User	DSA, ECDSA or RSA private key	Create
Export private key	N/A	User	DSA, ECDSA or RSA private key	Read
Random number generation	DRBG	User	Entropy input string, seed material	Read
			Internal state	Update
Message digest	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	User	None	N/A
	SHA3-224, SHA3-256, SHA3-384, SHA3-512			
Message authentication code (MAC)	HMAC	User	HMAC key	Read
	CMAC with AES	User	AES key	Read
	GMAC with AES	User	AES key	Read
Key encapsulation	RSA	User	RSA public and private keys	Read
Diffie-Hellman shared secret computation	KAS-FFC-SSC	User	Diffie-Hellman public and private keys	Read
			Shared secret	Create
Diffie-Hellman key generation using safe primes	Safe Primes Key Generation	User	Diffie-Hellman public and private keys	Create
EC Diffie-Hellman shared secret computation	KAS-ECC-SSC	User	EC Diffie-Hellman public and private keys	Read
			Shared secret	Create
Key derivation	KDF PBKDF	User	Password/passphrase	Read
			Derived key	Create
Network Protocol Services				
Transport Layer Security (TLS) network protocol v1.0, v1.1 and v1.2	Supported cipher suites in FIPS mode (see Appendix A for the complete list of valid cipher suites)	User	RSA, DSA or ECDSA public and private keys	Read
			TLS pre_master_secret, TLS master_secret, Diffie Hellman or EC Diffie Hellman public and private keys, AES or Triple-DES key, HMAC key	Create
TLS extensions	n/a	User	RSA, DSA or ECDSA public and private keys	Read

Service	Algorithm	Role	Keys/CSPs	Access
Certificate management	n/a	Crypto Officer	RSA, DSA or ECDSA public and private keys	Read
Other FIPS-related Services				
Show status	N/A	User	None	N/A
Zeroization	N/A	User	All CSPs	Zeroize
Self-tests	AES, Diffie-Hellman, DSA, EC Diffie-Hellman, ECDSA, DRBG, HMAC, RSA, SHS, Triple-DES	User	None	N/A
Module installation and configuration	N/A	Crypto Officer	None	N/A
Module initialization	N/A	Crypto Officer	None	N/A

Table 5: Services in FIPS mode of operation

Table 6 lists the services only available in non-FIPS mode of operation. The details of the non-approved cryptographic algorithms available in non-FIPS mode can be found in Table 9.

Service	Algorithm / Modes	Role	Keys	Access
Cryptographic Services				
Symmetric key generation	DRBG	User	Symmetric keys other than AES, Triple-DES and HMAC	Create
Symmetric encryption and decryption	Algorithms listed in Table 9	User	Symmetric key	Read
Asymmetric key generation	RSA, DSA and ECDSA restrictions and algorithms listed in Table 9	User	RSA, DSA or ECDSA public and private keys	Create
Digital signature generation and verification	RSA, DSA, ECDSA, message digest restrictions, and algorithms listed in Table 9	User	RSA, DSA or ECDSA public and private keys	Read
Message digest	Algorithms listed in Table 9	User	None	N/A
Message authentication code (MAC)	HMAC and CMAC restrictions, and algorithms listed in Table 9	User	HMAC key, two-key Triple-DES key	Read
RSA key encapsulation	RSA keys smaller than 2048 bits.	User	RSA key pair	Read

Service	Algorithm / Modes	Role	Keys	Access
Diffie-Hellman shared secret computation	Diffie-Hellman restrictions listed in Table 9	User	Diffie-Hellman public and private keys	Read
EC Diffie-Hellman shared secret computation	EC Diffie-Hellman restrictions listed in Table 9	User	EC Diffie-Hellman public and private keys	Read
Key derivation	KDF PBKDF using non-approved message digest.	User	Password/passphrase	Read
			Derived key	Create
Network Protocol Services				
Transport Layer Security (TLS) network protocol v1.0, v1.1 and v1.2	Non-supported cipher suites (see Appendix A for the complete list of valid cipher suites)	User	RSA, DSA or ECDSA public and private keys	Read
			TLS pre_master_secret, TLS master_secret, Diffie Hellman or EC Diffie Hellman public and private keys, AES or Triple-DES key, HMAC key	Create
Transport Layer Security (TLS) network protocol v1.3		User	RSA, DSA or ECDSA public and private keys	Read
			TLS pre_master_secret, TLS master_secret, Diffie Hellman or EC Diffie Hellman public and private keys, AES or Triple-DES key, HMAC key	Create

Table 6: Services in non-FIPS mode of operation

3.3 Operator Authentication

The module does not implement user authentication. The role of the user is implicitly assumed based on the service requested.

3.4 Algorithms

The module provides a generic C implementation of algorithms in all processor architectures, and specific implementations for the following processor architectures:

- For the Intel Xeon processor architecture:
  - use of AES-NI and SSSE3 instructions for AES implementations;
  - use of SSSE3 instructions for SHA implementations;
  - use of the CLMUL instruction set and strict assembler for GHASH that is used in GCM mode.
- For the ARMv8 processor architecture:
  - use of the Crypto Extensions instructions for AES and SHA implementations.

No additional implementations are provided for the z/15 processor architecture.

The module uses the most efficient implementation based on the processor’s capability. Notice that only one algorithm implementation can be executed in runtime.

Notice that for the Transport Layer Security (TLS), no parts of this protocol other than the key derivation function (SP800-135 TLS KDF), has been tested by the CAVP.

Table 7 lists the approved algorithms, the CAVP certificates, and other associated information of the cryptographic implementations in FIPS mode. Please refer to Appendix B for more detailed information about the algorithm implementations tested for each CAVP certificate.

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
AES	CFB8	128, 192, 256	Data Encryption and Decryption	FIPS197, SP800-38A	#A410 #A414 #A415
	CBC	128, 192, 256	Data Encryption and Decryption	FIPS197, SP800-38A	#A408 #A411 #A412 #A417
	CCM	128, 256	Data Encryption and Decryption	SP800-38C	#A411 #A417
	CMAC	128	MAC Generation and Verification	SP800-38B	#A408 #A411 #A412
	GCM	128, 256	Data Encryption and Decryption	SP800-38D	#A408 #A411 #A412 #A417
	GMAC	128, 256	Data Encryption and Decryption	SP800-38D	#A408
	XTS	128, 256	Data Encryption and Decryption	SP800-38E	#A416
DRBG	CTR_DRBG: AES-256 without DF, without PR	N/A	Deterministic Random Bit Generation	SP800-90A	#A408
DSA		L=2048, N=224 L=2048, N=256 L=3072, N=256	Key Pair Generation	FIPS186-4	#A408
	SHA-384	L=2048, N=224 L=2048, N=256 L=3072, N=256	Domain Parameter Generation		
	SHA-224, SHA-256, SHA-384, SHA-512	L=2048, N=224	Digital Signature Generation		
	SHA-256, SHA-384, SHA-512	L=2048, N=256 L=3072, N=256			
	SHA-384	L=2048, N=224 L=2048, N=256 L=3072, N=256	Domain Parameter Verification		

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	L=2048, N=224	Digital Signature Verification		
	SHA-1, SHA-256, SHA-384, SHA-512	L=2048, N=256 L=3072, N=256			
ECDSA		P-256, P-384, P-521	Key Pair Generation and Public Key Verification	FIPS186-4	#A408
	SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Digital Signature Generation		
	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Digital Signature Verification		
HMAC	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message authentication code	FIPS198-1	#A408 #A412
	SHA-1, SHA-224, SHA-256	112 or greater	Message authentication code	FIPS198-1	#A417
KAS-ECC-SSC	ECC Ephemeral Unified Scheme	P-256, P-384, P521	EC Diffie-Hellman Key Agreement	SP800-56Arev3	#A766
KAS-FFC-SSC	DhEphem Scheme with safe prime groups	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Diffie-Hellman Key Agreement	SP800-56Arev3	#A766
Safe Primes Key Generation	Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	2048, 3072, 4096, 6144, 8192	Diffie-Hellman Key Agreement	SP800-56Arev3	#A766

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
KDF PBKDF	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512		Key Derivation	SP800-132	Vendor affirmed[1]
KDF TLS	TLS v1.0, v1.1, v1.2 with SHA-256, SHA-384		Key Derivation	SP800-135	CVL. #A408
RSA		2048, 3072, 4096	Key Pair Generation	FIPS186-4	#A408
	PKCS#1v1.5: SHA-224, SHA-256, SHA-384, SHA-512	2048, 3072, 4096	Digital Signature Generation		
	PKCS#1v1.5: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	2048, 3072, 4096	Digital Signature Verification		
SHS	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest	FIPS180-4	#A408 #A412
	SHA-1, SHA-224, SHA-256	N/A	Message Digest	FIPS180-4	#A417
SHA-3	SHA3-224, SHA3-256, SHA3-384, SHA3-512	N/A	Message Digest	FIPS202	#A409 #A413
Triple-DES	CBC	192 (two-key Triple-DES)	Data Decryption	SP800-67 SP800-38A	#A408
		192 (three-key Triple-DES)	Data Encryption and Decryption	SP800-67 SP800-38A	#A408
KTS	AES CCM	128, 256	Key Wrapping and unwrapping as part of the cipher suites in the TLS protocol	SP800-38F	#A411 #A417
	AES GCM	128, 256			#A408 #A411 #A412 #A417
	AES CBC and HMAC	128, 256			#A408 #A411 #A412 #A417
	Triple DES CBC and HMAC	192			#A408 #A412 #A417

Table 7: Approved Cryptographic Algorithms

1 The PBKDF algorithm was tested with CAVP certificate #A408, but is considered vendor affirmed as the module doesn't implement the required known answer test (KAT) during selftests.

3.5 Allowed Algorithms

Table 8 describes the non-approved but allowed algorithms in FIPS mode:

Algorithm	Use
RSA Key Encapsulation with Encryption and Decryption Primitives with keys equal or larger than 2048 bits up to 15360 or more.	Key Establishment; allowed per [FIPS140-2_IG] D.9
MD5	Pseudo-random function (PRF) in TLS v1.0 and v1.1; allowed per [SP800-52rev2]
NDRNG	The module obtains the entropy data from a NDRNG to seed the DRBG.

Table 8: Non-Approved but Allowed Algorithms

3.5.1 Non-Approved Algorithms

Table 9 shows the non-Approved cryptographic algorithms implemented in the module that are only available in non-FIPS mode.

Algorithm	Use
Blowfish, Camellia, CAST, ChaCha20, DES, GOST, RC2, RC4, Salsa20, SEED, Serpent, Twofish	Data Encryption and Decryption.
2-key Triple-DES	Data Encryption.
Chacha20 and Poly1305	Authenticated Data Encryption and Decryption.
GOST, MD2, MD4, MD5, RMD160, STREEBOG	Message Digest.
UMAC, GMAC	Message Authentication Code.
HMAC with GOST, MD2, MD5, RMD160, STREEBOG	Message Authentication Code.
HMAC with less than 112-bit keys	Message Authentication Code.
SRP	Key Agreement.
SHA-1	Digital Signature Generation.
DSA with keys smaller than 2048 bits or greater than 3072 bits.	Key Pair Generation, Domain Parameter Generation.
DSA with keys smaller than 2048 bits or greater than 3072 bits. DSA with L=2048, N=256 or L=3072, N=256 and using SHA-1 or SHA-224.	Digital Signature Generation.
DSA with L=2048, N=256 or L=3072, N=256 and using SHA-224.	Digital Signature Verification.
DSA with keys smaller than 1024 bits or greater than 3072 bits.	Domain Parameter Verification, Digital Signature Verification.
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 with keys smaller than 2048 bits.	Key Encapsulation.

Algorithm	Use
ECDSA with P-192 and P-224 curves, K curves, B curves and non-NIST curves.	Key Pair Generation, Public Key Verification, Digital Signature Generation and Verification.
Diffie-Hellman with keys generated with domain parameters other than safe primes.	Key Agreement, Shared Secret computation.
Elliptic curve 25519.	Key Pair Generation, Public Key Verification, Digital Signature Generation and Verification.
EC Diffie-Hellman with P-192 and P-224 curves, K curves, B curves and non-NIST curves.	Key Agreement, Shared Secret computation.
PBKDF with non-approved message digest algorithms.	Key Derivation.
Yarrow	Random Number Generator

Table 9: Non-Approved Cryptographic Algorithms

4 Physical Security

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

5 Operational Environment

This module operates in a modifiable operational environment per the FIPS 140-2 level 1 specifications. The module runs on a commercially available general-purpose operating system executing on the hardware specified in Table 3.

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

- SLES
- SLES for SAP
- SLED
- SLE Micro

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

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.

5.1 Policy

The operating system is restricted to a single operator; concurrent operators are explicitly excluded.

The application that requests cryptographic services is the single user of the module.

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

6 Cryptographic Key Management

Table 10 summarizes the Critical Security Parameters (CSPs) that are used by the cryptographic services implemented in the module:

Name	Generation	Entry and Output	Zeroization
AES keys	Symmetric keys can be generated using the SP800-90A DRBG. Key material is also generated during Diffie-Hellman or EC Diffie-Hellman key agreement.	Keys are passed into the module via API input parameters in plaintext. Keys generated are returned to the calling application via an API output parameter.	`gnutls_cipher_deinit()` `gnutls_aead_cipher_deinit()`
Triple-DES keys			`gnutls_cipher_deinit()`
HMAC keys			`gnutls_hmac_deinit()`

RSA public and private keys	Public and private keys are generated using the FIPS 186-4 key generation method; the random value used in key generation is obtained from the SP800-90A DRBG.	Keys are passed into the module via API input parameters in plaintext.
		Keys are passed out of the module via API output parameters in plaintext.
			`gnutls_privkey_deinit()`
			`gnutls_x509_privkey_deinit()`
			`gnutls_rsa_params_deinit()`

DSA public and private keys			
ECDSA public and private keys			
Diffie-Hellman public and private keys	Public and private keys are generating using the SP800-56Arev3 Safe Primes key generation method, random values are obtained from the SP800-90A DRBG.	The key is passed into the module via API input parameters in plaintext. Keys are passed out of the module via API output parameters in plaintext.	`gnutls_dh_params_deinit()` `gnutls_pk_params_clear()`
EC Diffie-Hellman public and private keys	Public and private keys are generated using the SP800-56Arev3 key generation method, random values are obtained from the SP800-90A DRBG.	The key is passed into the module via API input parameters in plaintext. Keys are passed out of the module via API output parameters in plaintext.	`gnutls_pk_params_clear()`
Shared secret	Generated during the Diffie-Hellman or EC Diffie-Hellman key agreement and shared secret computation.	N/A	`zeroize_key()`
PBKDF password or passphrase	Not Applicable. Key material is entered via API parameters.	The key is passed into the module via API input parameters in plaintext.	Internal PBKDF state is zeroized automatically when function returns.
Derived key	Generated during the TLS KDF	N/A	Internal state is zeroized automatically when function returns.
	Generated during the PBKDF	Keys are passed out of the module via API output parameters in plaintext.	
Entropy input string and seed material	Obtained from NDRNG	N/A	`gnutls_global_deinit()`
DRBG internal state (V value, key)	Derived from entropy input as defined in SP800-90A	N/A	`gnutls_global_deinit()`

Name	Generation	Entry and Output	Zeroization
TLS pre_master_secret	Generated from the SP800-90A DRBG when module acts as a TLS client, for RSA cipher suites.	Received from TLS client (network), wrapped with TLS server's RSA public key, when module acts as a TLS server with RSA cipher suites.	`gnutls_deinit()`
	Generated during key agreement for Diffie-Hellman or EC Diffie-Hellman cipher suites.	N/A	
TLS master_secret	Derived from TLS pre_master_secret using TLS KDF.	N/A	`gnutls_deinit()`

Table 10: Life cycle of Keys or CSPs

The following sections describe how CSPs, in particular cryptographic keys, are managed during its life cycle.

6.1 Random Number Generation

The module employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90A] for the creation of seeds for asymmetric keys, and server and client random numbers for the TLS protocol. In addition, the module provides a Random Number Generation service to calling applications.

The DRBG supports the CTR_DRBG with AES-256, without derivation function and without prediction resistance.

The module uses a Non-Deterministic Random Number Generator (NDRNG), 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 module uses the getrandom() system call to access the output of NDRNG and seed the DRBG with 384 bits. The NDRNG provides at least 128 bits of entropy to the DRBG during initialization (seed) and reseeding (reseed).

The Linux kernel performs conditional self-tests on the output of NDRNG to ensure that consecutive random numbers do not repeat. The module performs the DRBG health tests as defined in section 11.3 of [SP800-90A].

6.2 Key/CSP Generation

The module provides an SP800-90A-compliant Deterministic Random Bit Generator (DRBG) for creation of key components of symmetric keys, key components of asymmetric keys, and random number generation.

The key generation methods implemented in the module for Approved services in FIPS mode is compliant with [SP800-133] (vendor affirmed).

- For generating AES, Triple-DES and HMAC keys, the module provides a symmetric key generation service compliant with [SP800-133].
- For generating RSA, DSA and ECDSA keys, the module implements asymmetric key generation services compliant with [FIPS186-4]. A seed (i.e. the random value) used in asymmetric key generation is directly obtained from the [SP800-90A] DRBG.
- 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], and generates keys using safe primes defined in RFC7919 and RFC3526, as described in the next section.

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

6.3 Key Agreement / Key Transport / Key Derivation

The module provides Diffie-Hellman and EC Diffie-Hellman key agreement schemes compliant with SP800-56Arev3, and used as part of the TLS protocol key exchange in accordance with scenario X1 (2) of IG D.8; that is, the shared secret computation (KAS-FFC-SSC and KAS-ECC-SSC) followed by the derivation of the keying material using SP800-135 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 of safe primes, and no other part of the IKE.

- 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 in CBC mode and HMAC, used by the TLS protocol cipher suites with 128-bit or 256-bit keys.
- Key wrapping using Triple-DES in CBC mode and HMAC, used by the TLS protocol cipher suites with 192-bit keys.
- RSA key encapsulation using private key encryption and public key decryption (also used as part of the TLS protocol key exchange).

According to Table 2: Comparable strengths in [SP 800-57], the key sizes of AES, RSA, Diffie-Hellman and EC Diffie-Hellman provide the following security strength in FIPS 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.
- Triple-DES key wrapping using HMAC provides 112 bits of encryption strength.
- RSA key wrapping[2] provides between 112 and 256 bits of encryption strength.
- Diffie-Hellman key agreement provides between 112 and 200 bits of encryption strength.
- EC Diffie-Hellman key agreement provides between 128 and 256 bits of encryption strength.

2 "Key wrapping” is used instead of “Key encapsulation” to show how the algorithm will appear in the certificate per IG G.13.

Note: As the module supports RSA key pairs greater than 2048 bits up to 15360 bits or more, an strength of 256 bits is claimed for RSA key encapsulation.

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

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

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

6.4 Key/CSP Entry and Output

The module does not support manual key entry or intermediate key generation key output. Instead, the module does provide services to import and export keys. Keys are provided to the module via API input parameters in plaintext form and output via API output parameters in plaintext form. This is allowed by [FIPS140-2_IG] IG 7.7, according to the “CM Software to/from App Software via GPC INT Path” entry on the Key Establishment Table.

6.5 Key/CSP Storage

Symmetric keys, HMAC 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 keys. The keys and CSPs are stored as plaintext in the RAM. The only exception is the HMAC key used for the Integrity Test, which is stored in the module and relies on the operating system for protection.

6.6 Key/CSP Zeroization

The memory occupied by keys is allocated by regular memory allocation operating system calls. The application is responsible for calling the appropriate zeroization functions provided in the module's API and listed in Table 10. Using the functions provided in this table will zeroize the keys and CSPs stored in the TLS protocol internal state and also invoke the corresponding API functions listed in Table 10 to zeroize keys and CSPs. The zeroization functions overwrite the memory occupied by keys with “zeros” and deallocate the memory with the regular memory deallocation operating system call.

7 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The test platforms as shown in Table 3 are compliant to 47 CFR FCC Part 15, Subpart B, Class A (Business use).

8 Self Tests

8.1 Power-Up Tests

The module performs power-up tests when the module is loaded into memory, without operator intervention. Power-up tests ensure that the module is not corrupted and that the cryptographic algorithms work as expected.

While the module is executing the power-up tests, services are not available, and input and output are inhibited. The module is not available for use by the calling application until the power-up tests are completed successfully.

If any power-up test fails, the module returns the error code listed in section 9.3 and displays the specific error message associated with the returned error code, and then enters the Error state. Subsequent calls to the module will also fail; no further cryptographic operations are possible. If the power-up tests complete successfully, the module returns the control to the calling application and is ready to accept cryptographic operation service requests.

8.1.1 Integrity Tests

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

8.1.2 Cryptographic Algorithm Tests

The module performs self-tests on all FIPS-Approved cryptographic algorithms supported in the Approved mode of operation, using the Known Answer Tests (KAT) and Pair-wise Consistency Tests (PCT) shown in the following table:

Algorithm	Power-Up Tests
AES	KAT AES CBC mode with 128-bit key, encryption and decryption (separately tested) KAT AES GCM mode with 256-bit key, encryption and decryption (separately tested) KAT AES XTS mode with 256-bit keys, encryption and decryption (separately tested)
Diffie-Hellman	Primitive “Z” Computation KAT
DRBG	KAT CTR_DRBG with AES with 256-bit keys without DF, without PR.
DSA	KAT DSA with L=2048, N=256 using SHA-256, signature generation and verification (separately tested). PCT DSA with L=3072, N=256 using SHA-256.
EC Diffie-Hellman	Primitive “Z” Computation KAT
ECDSA	KAT ECDSA with P-256 using SHA-256, P-384 using SHA-256, and P-521 using SHA-512, signature generation and verification (separately tested). PCT ECDSA with P-256 using SHA-256, P-384 using SHA-384, and P-521 using SHA-512.
HMAC	KAT HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512.

Algorithm	Power-Up Tests
RSA	KAT RSA with 2048-bit key using SHA-256, signature generation and verification (separately tested). KAT RSA with 2048-bit key, public key encryption and private key decryption (separately tested). PCT RSA with 3072-bit key using SHA-256.
SHA-3	KAT SHA3-224, SHA3-256, SHA3-384 and SHA3-512.
SHS	Covered by HMAC KATs.
TLS KDF	KAT with SHA-256
Triple-DES	KAT Triple-DES CBC mode, encryption and decryption (separately tested).

Table 11: Self-Tests

For the KAT, the module calculates the result and compares it with the known value. If the answer does not match the known answer, the KAT fails and the module enters the Error state. For the PCT, if the signature generation or verification fails, the module enters the Error state.

As described in section 3.4, only one AES or SHA implementation from libnettle library written in C language or using the support from AES-NI or SSSE3 instructions is available at run-time. The KATs cover different implementations dependent on the implementations availability in the operating environment.

8.2 On-Demand Self-Tests

On-Demand self-tests can be invoked by powering-off and reloading the module which cause the module to run the power-up tests again.

8.3 Conditional Tests

The module performs conditional tests on the cryptographic algorithms, using the Pair-wise Consistency Tests (PCT) shown in the following table. If the conditional test 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	Conditional Tests
DSA key generation	PCT: signature generation and verification using SHA-256.
ECDSA key generation	PCT: signature generation and verification using SHA-256.
RSA key generation	PCT: signature generation and verification using SHA-256. PCT: public encryption and private decryption.

Table 12: Conditional Tests

9 Guidance

9.1 Crypto Officer Guidance

The binaries of the module are contained in the RPM packages for delivery. The Crypto Officer shall follow this Security Policy to configure the operational environment and install the module to be operated as a FIPS 140-2 validated module.

The following RPM packages contain the FIPS validated module:

Processor Architecture	RPM Packages
Intel 64-bit	libgnutls30-3.6.7-14.4.1.x86_64.rpm libnettle6-3.4.1-4.12.1.x86_64.rpm libhogweed4-3.4.1-4.12.27.x86_64.rpm libgmp10-6.1.2-4.3.1.x86_64.rpm
IBM z15	libgnutls30-3.6.7-14.4.1.s390x.rpm libnettle6-3.4.1-4.12.1.s390x.rpm libhogweed4-3.4.1-4.12.27.s390x.rpm libgmp10-6.1.2-4.3.1.s390x.rpm
ARMv8 64-bit	libgnutls30-3.6.7-14.4.1.aarch64.rpm libnettle6-3.4.1-4.12.1.aarch64.rpm libhogweed4-3.4.1-4.12.27.aarch64.rpm libgmp10-6.1.2-4.3.1.aarch64.rpm

Table 13: RPM packages

9.1.1 Module Installation

The Crypto Officer can install the RPM packages containing the module as listed in Table 13 using the zypper tool. The integrity of the RPM package is automatically verified during the installation, and the Crypto Officer shall not install the RPM package if there is any integrity error.

9.1.2 Operating Environment Configuration

The operating environment needs to be configured to support FIPS, so the following steps shall be performed with the root privilege:

1. Install the dracut-fips RPM package:

```
# zypper install dracut-fips
```

2. Recreate the INITRAMFS image:

```
# dracut -f
```

3. After regenerating the initrd, the Crypto Officer has to append the following parameter in the /etc/default/grub configuration file in the GRUB_CMDLINE_LINUX_DEFAULT line:

```
fips=1
```

4. After editing the configuration file, please run the following command to change the setting in the boot loader:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

If /boot or /boot/efi resides on a separate partition, the kernel parameter boot=<partition of /boot or /boot/efi> must be supplied. The partition can be identified with the command "df /boot" or "df /boot/efi" respectively. 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 in the aforementioned grub file:

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

5. Reboot to apply these settings.

Now, the operating environment is configured to support FIPS operation. The Crypto Officer should check the existence of the file /proc/sys/crypto/fips_enabled, and verify it contains a numeric value “1”. If the file does not exist or does not contain “1”, the operating environment is not configured to support FIPS and the module will not operate as a FIPS validated module properly.

9.2 User Guidance

In order to run in FIPS mode, the module must be operated using the FIPS Approved services, with their corresponding FIPS Approved and FIPS allowed cryptographic algorithms provided in this Security Policy (see section 3.2). In addition, key sizes must comply with [SP800-131A].

9.2.1 TLS

The TLS protocol implementation provides both server and client sides. In order to operate in FIPS mode, digital certificates used for server and client authentication shall comply with the restrictions of key size and message digest algorithms imposed by [SP800-131A]. In addition, as required also 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 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.

9.2.2 Restrictions on environment variables and API functions

The module cannot use the following environment variables:

- GNUTLS_NO_EXPLICIT_INIT
- GNUTLS_SKIP_FIPS_INTEGRITY_CHECKS

The module can only be used with the cryptographic algorithms provided. Therefore, the following API functions are forbidden in FIPS mode of operation:

- gnutls_crypto_register_cipher
- gnutls_crypto_register_aead_cipher
- gnutls_crypto_register_mac
- gnutls_crypto_register_digest
- gnutls_privkey_import_ext4

9.2.3 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.

To meet the requirement stated in IG A.9, the module implements a check to ensure that the two AES keys used in AES XTS mode are not identical.

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

9.2.4 AES GCM IV

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 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.

The AES GCM IV generation is in compliance with the [RFC5288] and shall only be used for the TLS protocol version 1.2 to be compliant with [FIPS140-2_IG] IG A.5, provision 1 (“TLS protocol IV generation”); thus, the module is compliant with section 3.3.1 of [SP800-52rev2].

When a GCM IV is used for decryption, the responsibility for the IV generation lies with the party that performs the AES GCM encryption and therefore there is no restriction on the IV generation.

9.2.5 Triple-DES encryption

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

[SP800-67] imposes a restriction on the number of 64-bit block encryptions performed under the same three-key Triple-DES key.

When the three-key Triple-DES is generated as part of a recognized IETF protocol, the module is limited to 2^{20} 64-bit data block encryptions. This scenario occurs in the following protocols:

- Transport Layer Security (TLS) versions 1.1 and 1.2, conformant with [RFC5246]
- Secure Shell (SSH) protocol, conformant with [RFC4253]
- Internet Key Exchange (IKE) versions 1 and 2, conformant with [RFC7296]

In any other scenario, the module cannot perform more than 2^{16} 64-bit data block encryptions.

The user is responsible for ensuring the module’s compliance with this requirement.

9.2.6 Key derivation using SP800-132 PBKDF

The module provides password-based key derivation (PBKDF), compliant with SP800-132. 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 to [SP800-132] and IG D.6, the following requirements shall be met.

- Derived keys shall only be used in storage applications. The Master Key (MK) shall not be used for other purposes. The length of the MK or DPK shall be of 112 bits or more.
- A portion of the salt, with a length of at least 128 bits, shall be generated randomly using the SP800-90A 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.

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

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

9.3 Handling Self-Test Errors

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

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

Table 14: Error Events, Error Codes and Error Messages

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

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

10 Mitigation of Other Attacks

The module does not offer mitigation of other attacks.

Appendix A - TLS Cipher Suites

The module supports the following cipher suites for the TLS protocol versions 1.0, 1.1 and 1.2, 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	Reference
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_RSA_WITH_AES_128_CBC_SHA	RFC3268
TLS_DH_DSS_WITH_AES_128_CBC_SHA	RFC3268
TLS_DH_RSA_WITH_AES_128_CBC_SHA	RFC3268
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	RFC3268
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	RFC3268
TLS_DH_anon_WITH_AES_128_CBC_SHA	RFC3268
TLS_RSA_WITH_AES_256_CBC_SHA	RFC3268
TLS_DH_DSS_WITH_AES_256_CBC_SHA	RFC3268
TLS_DH_RSA_WITH_AES_256_CBC_SHA	RFC3268
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	RFC3268
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	RFC3268
TLS_DH_anon_WITH_AES_256_CBC_SHA	RFC3268
TLS_RSA_WITH_AES_128_CBC_SHA256	RFC5246
TLS_RSA_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DH_DSS_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DH_RSA_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DH_DSS_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DH_RSA_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DH_anon_WITH_AES_128_CBC_SHA256	RFC5246

Cipher Suite	Reference
TLS_DH_anon_WITH_AES_256_CBC_SHA256	RFC5246
TLS_PSK_WITH_3DES_EDE_CBC_SHA	RFC4279
TLS_PSK_WITH_AES_128_CBC_SHA	RFC4279
TLS_PSK_WITH_AES_256_CBC_SHA	RFC4279
TLS_RSA_WITH_AES_128_GCM_SHA256	RFC5288
TLS_RSA_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DH_RSA_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DH_RSA_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DH_DSS_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DH_DSS_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DH_anon_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DH_anon_WITH_AES_256_GCM_SHA384	RFC5288
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA	RFC4492
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA	RFC4492
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA	RFC4492
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	RFC4492
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	RFC4492
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	RFC4492
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA	RFC4492
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA	RFC4492
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA	RFC4492
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	RFC4492
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	RFC4492
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	RFC4492
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA	RFC4492
TLS_ECDH_anon_WITH_AES_128_CBC_SHA	RFC4492
TLS_ECDH_anon_WITH_AES_256_CBC_SHA	RFC4492
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	RFC5289
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	RFC5289

Cipher Suite	Reference
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256	RFC5289
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384	RFC5289
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	RFC5289
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	RFC5289
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	RFC5289
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384	RFC5289
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	RFC5289
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	RFC5289
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	RFC5289
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384	RFC5289
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	RFC5289
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	RFC5289
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	RFC5289
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	RFC5289
TLS_RSA_WITH_AES_128_CCM	RFC6655
TLS_RSA_WITH_AES_256_CCM	RFC6655
TLS_DHE_RSA_WITH_AES_128_CCM	RFC6655
TLS_DHE_RSA_WITH_AES_256_CCM	RFC6655
TLS_RSA_WITH_AES_128_CCM_8	RFC6655
TLS_RSA_WITH_AES_256_CCM_8	RFC6655
TLS_DHE_RSA_WITH_AES_128_CCM_8	RFC6655
TLS_DHE_RSA_WITH_AES_256_CCM_8	RFC6655

Table 15: TLS Cipher Suites

Appendix B - CAVP certificates

The tables below show the certificates obtained from the CAVP for all the target platforms included in Table 3. The CAVP certificates validate all algorithm implementations used as approved or allowed security functions in FIPS mode of operation. The tables include the certificate number, the label used in the CAVP certificate for reference and a description of the algorithm implementation.

Cert#	CAVP Label	Algorithm Implementation
A408	Generic_C	AES, DRBG, DSA, ECDSA, HMAC, KAS-ECC, KAS-FFC, KDF TLS, PBKDF, RSA, SHA and Triple-DES implementations using generic C.
A409	C_SHA3	SHA-3 implementation using generic C.
A410	C_CFB8	AES-CFB8 implementation using generic C.
A411	AESNI	AES implementation using AESNI instructions.
A412	SSSE3	AES and SHA implementations using SSSE3 instruction.
A413	SSSE3_SHA3	SHA-3 implementation using SSSE3 instruction.
A414	SSSE3_CFB8_CMAC	AES-CFB8 implementation using SSS3 instruction.
A415	AESNI_CFB8_CMAC	AES-CFB8 implementation using AESNI instruction.
A416	Generic_C_XTS	AES-XTS implementation in generic C language.
A766	SP800 56A rev 3	SP800-56A rev 3 compliant implementation.

Table 16: CAVP certificates for the Intel Xeon processor

Cert#	CAVP Label	Algorithm Implementation
A408	Generic_C	AES, DRBG, DSA, ECDSA, HMAC, KAS-ECC, KAS-FFC, KDF TLS, PBKDF, RSA, SHA and Triple-DES implementations using generic C.
A409	C_SHA3	SHA-3 implementation using generic C.
A410	C_CFB8	C implementation of AES in CFB8 mode.
A416	Generic_C_XTS	AES-XTS implementation in generic C language.
A766	SP800 56A rev 3	SP800-56A rev 3 compliant implementation.

Table 17: CAVP certificates for the IBM z15 processor

Cert#	CAVP Label	Algorithm Implementation
A408	Generic_C	AES, DRBG, DSA, ECDSA, HMAC, KAS-ECC, KAS-FFC, KDF TLS, PBKDF, RSA, SHA and Triple-DES implementations using generic C.
A409	C_SHA3	SHA-3 implementation using generic C.
A410	C_CFB8	C implementation of AES in CFB8 mode.
A416	Generic_C_XTS	AES-XTS implementation in generic C language.
A417	CE	AES and SHA implementations using Crypto Extension instructions.
A766	SP800 56A rev 3	SP800-56A rev 3 compliant implementation.

Table 18: CAVP certificates for the ARMv8 processor

Appendix C - Glossary and Abbreviations

AES	Advanced Encryption Specification
AES_NI	Intel® Advanced Encryption Standard (AES) New Instructions
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CCM	Counter with Cipher Block Chaining Message Authentication Code
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
FIPS	Federal Information Processing Standards Publication
GCM	Galois Counter Mode
HMAC	Hash Message Authentication Code
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
PKCS	Public Key Cryptography Standards
RNG	Random Number Generator
RPM	Red hat Package Manager
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
TDES	Triple-DES
XTS	XEX Tweakable Block Cipher with Ciphertext Stealing

Appendix D - References

FIPS 140-2	**FIPS PUB 140-2 - Security Requirements for Cryptographic Modules** https://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
FIPS 140-2_IG	**Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program** December 3, 2019 https://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf
FIPS180-4	**Secure Hash Standard (SHS)** https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
FIPS186-4	**Digital Signature Standard (DSS)** https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS197	**Advanced Encryption Standard** https://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
FIPS198-1	**The Keyed Hash Message Authentication Code (HMAC)** https://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
FIPS202	**SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions** https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
PKCS#1	**Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1** https://www.ietf.org/rfc/rfc3447.txt
RFC2246	**The TLS Protocol Version 1.0** https://www.ietf.org/rfc/rfc2246.txt
RFC3268	**Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)** https://www.ietf.org/rfc/rfc3268.txt
RFC4279	**Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)** https://www.ietf.org/rfc/rfc4279.txt
RFC4346	**The Transport Layer Security (TLS) Protocol Version 1.1** https://www.ietf.org/rfc/rfc4346.txt
RFC4492	**Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)** https://www.ietf.org/rfc/rfc4492.txt
RFC5116	**An Interface and Algorithms for Authenticated Encryption** https://www.ietf.org/rfc/rfc5116.txt
RFC5246	**The Transport Layer Security (TLS) Protocol Version 1.2** https://tools.ietf.org/html/rfc5246.txt
RFC5288	**AES Galois Counter Mode (GCM) Cipher Suites for TLS** https://tools.ietf.org/html/rfc5288.txt

RFC5487 **Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode**
https://tools.ietf.org/html/rfc5487.txt

RFC5489 **ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)**
https://tools.ietf.org/html/rfc5489.txt

RFC6655 **AES-CCM Cipher Suites for Transport Layer Security (TLS)**
https://tools.ietf.org/html/rfc6655.txt

RFC7251 **AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS**
https://tools.ietf.org/html/rfc7251.txt

RFC7296 **Internet Key Exchange Protocol Version 2 (IKEv2)**
https://tools.ietf.org/html/rfc7296

SP800-38A **NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques**
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**
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**
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**
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**
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38e.pdf

SP800-38F **NIST Special Publication 800-38F - Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping**
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf

SP800-52rev2 **NIST Special Publication 800-52 Revision 2 - Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations**
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf

SP800-56A **NIST Special Publication 800-56A Revision 3 - Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography**

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf

SP800-67	**NIST Special Publication 800-67 Revision 1 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher** https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-67r1.pdf
SP800-90A	**NIST Special Publication 800-90A Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
SP800-131A	**NIST Special Publication 800-131A Revision 1- Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths** https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
SP800-132	**NIST Special Publication 800-132 - Recommendation for Password-Based Key Derivation - Part 1: Storage Applications** https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf