SUSE Linux Enterprise GnuTLS Cryptographic Module

version 1.1

FIPS 140-3 Non-Proprietary Security Policy

Version 1.2

Last update: 2024-07-25

Prepared by:

atsec information security corporation

4516 Seton Center Parkway, Suite 250

Austin, TX 78759

www.atsec.com

1 Table of Contents

1 General

This document is the non-proprietary FIPS 140-3 Security Policy for version 1.1 of the SUSE Linux Enterprise GnuTLS Cryptographic Module. It has a one-to-one mapping to the [SP 800-140B] starting with section B.2.1 named “General” that maps to section 1 in this document and ending with section B.2.12 named “Mitigation of other attacks” that maps to section 12 in this document.

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

In preparing the Security Policy document, the laboratory formatted the vendor-supplied documentation for consolidation without altering the technical statements therein contained. The further refining of the Security Policy document was conducted iteratively throughout the conformance testing, wherein the Security Policy was submitted to the vendor, who would then edit, modify, and add technical contents. The vendor would also supply additional documentation, which the laboratory formatted into the existing Security Policy, and resubmitted to the vendor for their final editing.

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

ISO/IEC 24759 Section 6. [Number Below]	FIPS 140-3 Section Title	Security Level
1	General	1
2	Cryptographic Module Specification	1
3	Cryptographic Module Interfaces	1
4	Roles, Services, and Authentication	1
5	Software/Firmware Security	1
6	Operational Environment	1
7	Physical Security	N/A
8	Non-invasive Security	N/A
9	Sensitive Security Parameter Management	1
10	Self-tests	1
11	Life-cycle Assurance	1
12	Mitigation of Other Attacks	N/A
Overall		1

Table 1 - Security Levels

2 Cryptographic Module Specification

2.1 Module Embodiment

The SUSE Linux Enterprise GnuTLS Cryptographic Module (hereafter referred to as “the module”) is a Software multi-chip standalone cryptographic module.

2.2 Module Design, Components, Versions

The software block diagram below shows the cryptographic boundary of the module, and its interfaces with the operational environment.

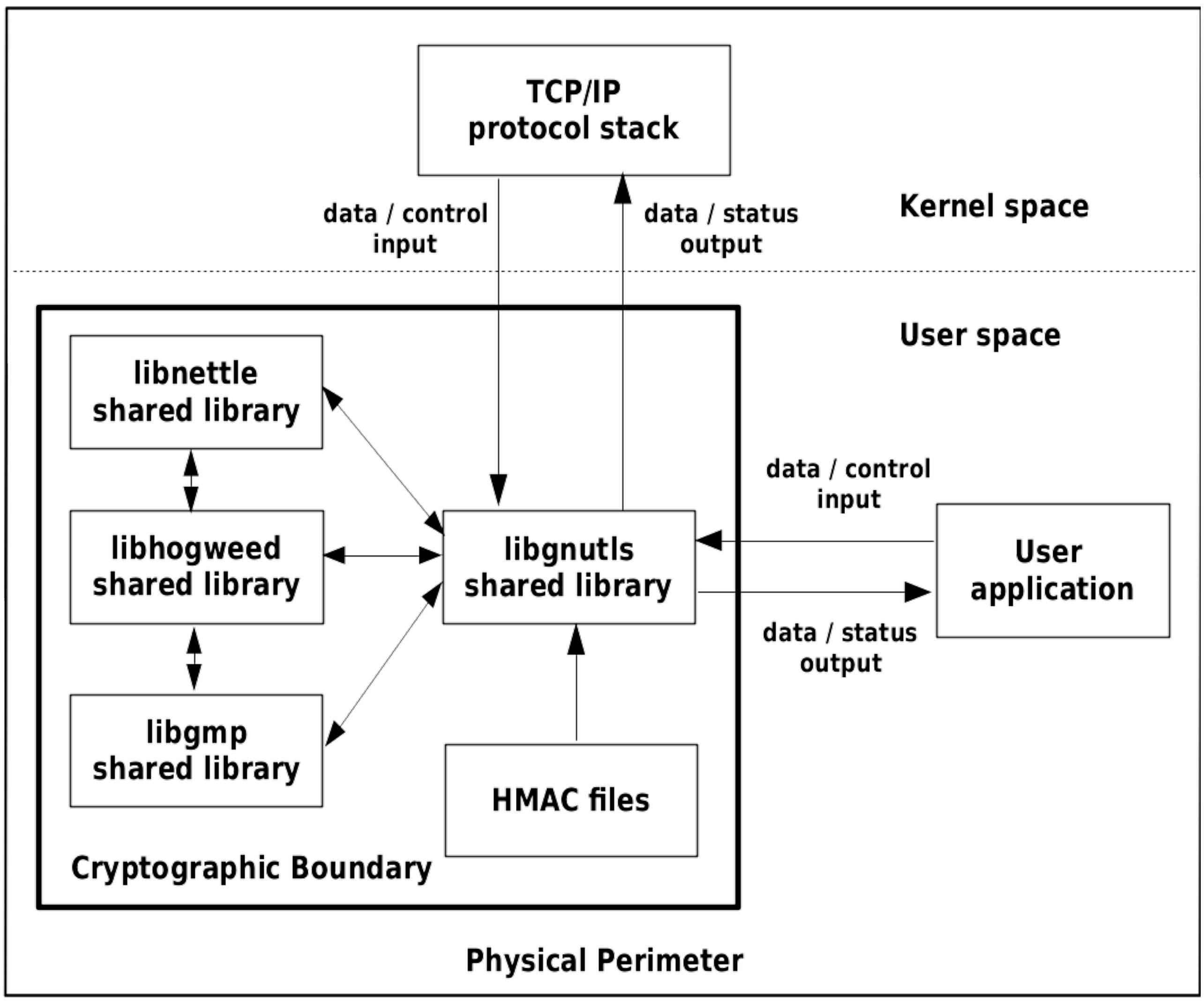

Figure 1 - Cryptographic Boundary

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

Components	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.8	Provides the cryptographic algorithm implementations, including AES, SHA, HMAC, RSA Digital Signature and ECDSA.
/usr/lib64/libhogweed.so.6	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-SHA2-256 values of their associated library for integrity check during the power-up.
/usr/lib64/.libnettle.so.8.hmac	
/usr/lib64/.libhogweed.so.6.hmac	
/usr/lib64/.libgmp.so.10.hmac	

Table 2 – Cryptographic Module Components

2.3 Modes of operation

When the module starts up successfully, after passing all the pre-operational and conditional cryptographic algorithms self-tests (CASTs), the module is operating in the approved mode of operation by default and can only be transitioned into the non-Approved mode by calling one of the non-Approved services listed in Table 11. The module switches between approved and non-approved mode based on the service requested. Please see section 4 for the details on service indicator provided by the module that identifies when an approved service is called.

2.4 Tested Operational Environments

The module has been tested on the following platforms with the corresponding module variants and configuration options:

#	Operating System	Hardware Platform	Processor	PAA/Acceleration
1	SUSE Linux Enterprise Server 15 SP4	Supermicro Super Server SYS-6019P-WTR	Intel® Xeon® Silver 4215R	With and without AES-NI (PAA)
2	SUSE Linux Enterprise Server 15 SP4	GIGABYTE R181-Z90-00	AMD EPYC™ 7371	With and without AES-NI (PAA)
3	SUSE Linux Enterprise Server 15 SP4	GIGABYTE G242-P32-QZ	ARM Ampere® Altra® Q80-30	With and without Crypto Extensions (PAA)

#	Operating System	Hardware Platform	Processor	PAA/Acceleration
4	SUSE Linux Enterprise Server 15 SP4	IBM z/15	z15	With and without CPACF (PAI)
5	SUSE Linux Enterprise Server 15 SP4 on PowerVM (VIOS 3.1.4.00)	IBM Power E1080 (9080-HEX)	Power10	With and without ISA (PAA)

Table 3 - Tested Operational Environments

2.5 Vendor-Affirmed Operational Environments

In addition to the platforms listed in Table 3, SUSE, LLC has also tested the module on the platforms in Table 4, and claims vendor affirmation on them.

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.

#	Operating System	Hardware platform	Processor	PAA/Acceleration
1	SUSE Linux Enterprise Server 15SP4	IBM LinuxONE III LT1	z15	With and without CPACF (PAI)
2	SUSE Linux Enterprise Micro 5.3	Supermicro Super Server SYS-6019P-WTR	Intel® Xeon® Silver 4215R	With and without AES-NI (PAA)
3	SUSE Linux Enterprise Micro 5.3	GIGABYTE R181-Z90-00	AMD EPYC™ 7371	With and without AES-NI (PAA)
4	SUSE Linux Enterprise Micro 5.3	GIGABYTE G242-P32-QZ	ARM Ampere® Altra® Q80-30	With and without Cryptography Extensions (PAA)
5	SUSE Linux Enterprise Micro 5.3	IBM z/15	z15	With and without CPACF (PAI)
6	SUSE Linux Enterprise Micro 5.3	IBM LinuxONE III LT1	z15	With and without CPACF (PAI)
7	SUSE Linux Enterprise Server for SAP 15SP4	Supermicro Super Server SYS-6019P-WTR	Intel® Xeon® Silver 4215R	With and without AES-NI (PAA)
8	SUSE Linux Enterprise Server for SAP 15SP4	GIGABYTE R181-Z90-00	AMD EPYC™ 7371	With and without AES-NI (PAA)
9	SUSE Linux Enterprise Server for SAP 15SP4	IBM Power E1080 (9080-HEX)	Power10	With and without ISA (PAA)

#	Operating System	Hardware platform	Processor	PAA/Acceleration
10	SUSE Linux Enterprise Base Container Image 15SP4	Supermicro Super Server SYS-6019P-WTR	Intel® Xeon® Silver 4215R	With and without AES-NI (PAA)
11	SUSE Linux Enterprise Base Container Image 15SP4	GIGABYTE R181-Z90-00	AMD EPYC™ 7371	With and without AES-NI (PAA)
12	SUSE Linux Enterprise Base Container Image 15SP4	GIGABYTE G242-P32-QZ	ARM Ampere® Altra® Q80-30	With and without Cryptography Extensions (PAA)
13	SUSE Linux Enterprise Base Container Image 15SP4	IBM z/15	z15	With and without CPACF (PAI)
14	SUSE Linux Enterprise Base Container Image 15SP4	IBM LinuxONE III LT1	z15	With and without CPACF (PAI)
15	SUSE Linux Enterprise Base Container Image 15SP4	IBM Power E1080 (9080-HEX)	Power10	With and without ISA (PAA)
16	SUSE Linux Enterprise Desktop 15SP4	Supermicro Super Server SYS-6019P-WTR	Intel® Xeon® Silver 4215R	With and without AES-NI (PAA)
17	SUSE Linux Enterprise Desktop 15SP4	GIGABYTE R181-Z90-00	AMD EPYC™ 7371	With and without AES-NI (PAA)
18	SUSE Linux Enterprise Real Time 15SP4	Supermicro Super Server SYS-6019P-WTR	Intel® Xeon® Silver 4215R	With and without AES-NI (PAA)
19	SUSE Linux Enterprise Real Time 15SP4	GIGABYTE R181-Z90-00	AMD EPYC™ 7371	With and without AES-NI (PAA)

Table 4 - Vendor-Affirmed Operational Environments

2.6 Approved Algorithms

Table 5 lists all security functions of the module, including specific key strengths employed for approved services, and implemented modes of operation.

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

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Size(s)/Key Strength(s)	Use/Function
A2984, A2985, A2986, A2987, A2992, A2996, A2997, A3004, A3007	AES FIPS197, SP800-38A	CBC	128, 192, 256-bit keys with 128-256 bits of key strength	Symmetric encryption; Symmetric decryption
A2984, A2996, A3004, A3007	AES SP800-38C	CCM	128, 256-bit keys with 128 or 256 bits of key strength	Symmetric encryption; Symmetric decryption; Authenticated symmetric encryption; Authenticated symmetric decryption
A2989, A2990, A2995	AES FIPS197, SP800-38A	CFB8	128, 192, 256-bit keys with 128-256 bits of key strength	Symmetric encryption; Symmetric decryption
A2984, A2987, A2992, A2996, A3004	AES SP800-38B	CMAC	128, 256-bit keys with 128 or 256 bits of key strength	Message authentication code (MAC)
A2984, A2985, A2986, A2987, A2992, A2996, A2997, A3004, A3007	AES SP800-38D RFC5288 RFC8446	GCM	128, 256-bit keys with 128 or 256 bits of key strength	Symmetric encryption and decryption in the context of the Transport Layer Security (TLS) network protocol
A2992	AES SP800-38D	GMAC	128, 256-bit keys with 128 or 256 bits of key strength	Message authentication code (MAC)
A2993	AES SP800-38E	XTS	128, 256-bit keys with 128 or 256 bits of key strength	Symmetric encryption (for data storage); Symmetric decryption (for data storage)
Vendor Affirmed	CKG SP800-133rev2	Key pair generation (FIPS-186-4, SP800-56Arev3, SP800-90Arev1);	RSA: 2048, 3072, 4096-bit keys with 112, 128, 149 bits of key strength ECDH/ECDSA: P-256, P-384, P-521 elliptic curves with 128-256 bits of key strength Safe Primes: 2048, 3072, 4096, 6144, 8192-bit keys with 112-200 bits of key strength	Key pair generation

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Size(s)/Key Strength(s)	Use/Function
A2992	DRBG SP800-90Arev1	CTR_DRBG: AES-256 without DF, without PR	AES 256-bit key with 256 bits of key strength	Random number generation
A2992	ECDSA FIPS186-4	ECDSA KeyGen (B.4.2 Testing Candidates)	P-256, P-384, P-521 elliptic curves with 128-256 bits of key strength	Key pair generation
		ECDSA KeyVer	P-256, P-384, P-521 elliptic curves with 128-256 bits of key strength	Public key verification
		ECDSA SigGen (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	P-256, P-384, P-521 elliptic curves with 128-256 bits of key strength	Digital signature generation
		ECDSA SigVer (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	P-256, P-384, P-521 elliptic curves with 128-256 bits of key strength	Digital signature verification
A2987, A2992, A2998, A3007	HMAC FIPS198-1	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	112-524288 bits with 112-256 bits of security strength	Message authentication code (MAC)
A2992	KAS-ECC-SSC SP800-56Arev3	ECC Ephemeral Unified Scheme	P-256, P-384, P-521 elliptic curves keys with 128-256 bits of key strength	EC Diffie-Hellman shared secret computation; Transport Layer Security (TLS) network protocol
A2992	KAS-FFC-SSC SP800-56Arev3	Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	2048, 3072, 4096, 6144, 8192-bit keys with 112-200 bits of key strength	Diffie-Hellman shared secret computation; Transport Layer Security (TLS) network protocol

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Size(s)/Key Strength(s)	Use/Function
A2991	KDA HKDF SP800-56Cr1	HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512	HKDF derived key with 112 to 256 bits of key strength	HKDF key derivation Transport Layer Security (TLS) network protocol
A2992	KDF TLS v1.0/1.1 SP800-135rev1 (CVL)	SHA-1	TLS Derived key with 112 to 256 bits of key strength	TLS key derivation
A2992	TLS v1.2 KDF SP800-135rev1 RFC7627 (CVL)	SHA2-256, SHA2-384	TLS Derived key with 112 to 256 bits of key strength	TLS key derivation
A2984, A2996, A3004, A3007	AES CCM SP800-38C	KTS per IG D.G	128, 256-bit keys with 128 or 256 bits of key strength	Key wrapping; Key unwrapping
A2984, A2985, A2986, A2987, A2992, A2996, A2997, A3004, A3007	AES GCM SP800-38D	KTS per IG D.G	128, 256-bit keys with 128 or 256 bits of key strength	
AES A2984, A2985, A2986, A2987, A2992, A2996, A2997, A3004, A3007 HMAC A2987, A2992, A2998, A3007	AES CBC and HMAC SP800-38A, FIPS198-1	KTS per IG D.G	128, 256-bit keys with 128 or 256 bits of key strength	
A2992	PBKDF SP800-132	HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512	8-128 characters with password strength between 10^8 and 10^{128}	Password-based key derivation
A2992	RSA FIPS186-4	RSA KeyGen (B.3.2 Random Provable Primes)	2048-15360 bits keys with 112-256 bits of security strength	Key pair generation

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Size(s)/Key Strength(s)	Use/Function
		RSA SigGen (PKCS#1v1.5: SHA2-224, SHA2-256, SHA2-384, SHA2-512)	2048-15360 bits keys with 112-256 bits of security strength	Digital signature generation
		RSA SigGen (PSS: SHA2-256, SHA2-384, SHA2-512)	2048-15360 bits keys with 112-256 bits of security strength	
		RSA SigVer (PKCS#1v1.5: SHA2-224, SHA2-256, SHA2-384, SHA2-512)	2048-15360 bits keys with 112-256 bits of security strength	Digital signature verification
		RSA SigVer (PSS: SHA2-256, SHA2-384, SHA2-512)	2048-15360 bits keys with 112-256 bits of security strength	
A2992	Safe Primes Key Generation SP800-56Arev3	Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	2048, 3072, 4096, 6144, 8192-bit keys with 112-200 bits of key strength	Key pair generation
A2988, A2994	SHA-3 FIPS202 FIPS 140-3 IG C.C	SHA3-224, SHA3-256, SHA3-384, SHA3-512	N/A	Message digest
A2987, A2992, A2998, A3007	SHA FIPS180-4	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	N/A	Message digest

Table 5 - Approved Algorithms

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

The module does not implement non-approved algorithms that are allowed in the approved mode of operation.

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

Table 6 lists the non-approved algorithms that are allowed in the approved mode of operation with no security claimed. These algorithms are used by the approved services listed in Table 10.

Algorithm[1]	Caveat	Use/Function
MD5	Only allowed as the PRF in TLSv1.0 and v1.1 per IG 2.4.A	Message digest used in TLS v1.0/1.1 KDF only

Table 6 - Non-Approved Algorithms Allowed in the Approved Mode of Operation with No Security Claimed

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

Table 7 lists non-approved algorithms that are not allowed in the approved mode of operation. These algorithms are used by the non-approved services listed in Table 11.

Algorithm/Functions	Use/Function
AES GCM when not used in the context of the TLS protocol.	Symmetric encryption; Symmetric decryption
Blowfish	Symmetric encryption; Symmetric decryption
Camellia	Symmetric encryption; Symmetric decryption
CAST	Symmetric encryption; Symmetric decryption
ChaCha20	Symmetric encryption; Symmetric decryption
Chacha20 and Poly1305	Authenticated encryption; Authenticated decryption
CMAC with Triple-DES	Message authentication code (MAC)
DES	Symmetric encryption; Symmetric decryption
Diffie-Hellman with keys generated with domain parameters other than safe primes	Key agreement; Diffie-Hellman shared secret computation

[1] These algorithms do not claim any security and are not used to meet FIPS 140-3 requirements. Therefore, SSPs do not map to these algorithms.

Algorithm/Functions	Use/Function
DSA	Key pair generation; Domain parameter generation; Digital signature generation; Digital signature verification
ECDSA with curves not listed in Table 5.	Key pair generation; Public key verification; Digital signature generation; Digital signature verification
EC Diffie-Hellman with curves not listed in Table 5	Key agreement; EC Diffie-Hellman shared secret computation
GMAC	Message authentication code (MAC)
GOST	Symmetric encryption; Symmetric decryption; Message digest
HMAC with keys smaller than 112-bit	Message authentication code (MAC)
HMAC with GOST	Message authentication code (MAC)
MD2, MD4, MD5	Message digest; Message authentication code (MAC)
Non-supported cipher suites (not listed in Appendix A)	Transport Layer Security (TLS) Network Protocol
PBKDF with non-approved message digest algorithms	Password-based key derivation
RC2, RC4	Symmetric encryption; Symmetric decryption
RMD160	Message digest; Message authentication code (MAC)
RSA with keys smaller than 2048 bits or greater than 4096 bits	Key pair generation; Digital signature generation
RSA with keys smaller than 1024 bits or greater than 4096 bits	Digital signature verification
RSA encryption and decryption with any key sizes	Key encapsulation; Key un-encapsulation
Salsa20	Symmetric encryption; Symmetric decryption
SEED	Symmetric encryption; Symmetric decryption
Serpent	Symmetric encryption; Symmetric decryption
SHA-1	Digital signature generation
SRP	Key agreement
STREEBOG	Message digest; Message authentication code (MAC)
Triple-DES	Symmetric encryption; Symmetric decryption
Twofish	Symmetric encryption; Symmetric decryption
UMAC	Message authentication code (MAC)

Algorithm/Functions	Use/Function
Yarrow	Random number generation

Table 7 - Non-Approved Not Allowed in the Approved Mode of Operation

3 Cryptographic Module Ports and Interfaces

As a software-only 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 are interpreted to be the physical ports of the hardware platform on which the module runs. The following table shows the logical interfaces implemented in the module.

All data output via data output interface is inhibited when the module is performing pre-operational test conditional cryptographic algorithms self-tests or zeroization or when the module enters error state.

Logical Interface[2]	Data that passes over port/interface
Data Input	API input parameters, kernel I/O network or files on filesystem, TLS protocol input messages.
Data Output	API output parameters, kernel I/O network or files on filesystem, TLS protocol output messages.
Control Input	API function calls, API input parameters for control.
Status Output	API return codes, API output parameters for status output.

Table 8 - Ports and Interfaces

[2] The control output interface is omitted on purpose because the module does not implement it.

4 Roles, services, and authentication

4.1 Services

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

Role	Service	Input	Output
Crypto Officer (CO)	Authenticated symmetric encryption	Key, Plaintext, IV	Ciphertext, MAC tag
	Authenticated symmetric decryption	Key, Ciphertext, MAC tag	Plaintext
	Key pair generation	RSA key size, Diffie-Hellman Safe Prime or Elliptic Curve	Key pair
	Diffie-Hellman shared secret computation	Private key, public key from peer	Shared secret
	Digital signature generation	Message, private key, hash algorithm	Digital signature
	Digital signature verification	Signature, message, public key, hash algorithm	Verification result
	Domain parameter generation	Domain parameters input	Generated domain parameters
	EC Diffie-Hellman shared secret computation	Private key, public key from peer	Shared secret
	HKDF key derivation	Shared secret	HKDF derived key
	TLS key derivation	TLS Pre-master secret	Derived key
	Key agreement	Key pair	Shared secret
	Key encapsulation	Key to be encapsulated, Key encapsulating key	Encapsulated key
	Key un-encapsulation	Encapsulated key, Key encapsulating key	Unencapsulated key
	Key unwrapping	Wrapped key, Key unwrapping key	Unwrapped key
	Key wrapping	Key to be wrapped, Key wrapping key	Wrapped key
	Message authentication code (MAC)	Message, HMAC key or AES key	Message authentication code
	Message digest	Message	Digest of the message

Role	Service	Input	Output
	Password-based key derivation	Password or passphrase, salt, iteration count	PBKDF Derived key
	Public key verification	Key pair	Pass/fail
	Random number generation	Number of bits	Random number
	Self-tests	Module reset or API call	Result of self-test (pass/fail)
	Symmetric decryption	Key, Ciphertext	Plaintext
	Symmetric encryption	Key, Plaintext	Ciphertext
	Show module name and version	None	Name and version information
	Show status	N/A	Return codes and/or log messages
	Transport Layer Security (TLS) network protocol	Cipher-suites, Digital Certificate, Public and Private Keys, Application Data	Return codes and/or log messages, Application data
	Zeroization	Context containing SSPs	N/A

Table 9 - Roles, Service Commands, Input and Output

The module provides services to the users that assume one of the available roles. All services are shown in Table 10 and Table 11.

4.1.1 Approved Services

Table 10 lists the approved services. For each service, the table lists the associated cryptographic algorithm(s), the role to perform the service, the cryptographic keys or SSPs involved, and their access type(s). The following convention is used to specify access rights to an SSP:

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

The details of the approved cryptographic algorithms including the CAVP certificate numbers can be found in Table 5.

The “Indicator” column shows the service indicator API function that must be used to verify the service indicator after executing a service. The gnutls_fips140_get_operation_state() function indicates GNUTLS_FIPS140_OP_APPROVED whether the API invoked corresponds to an approved algorithm.

Service	Description	Approved Security Functions	Keys and/or SSPs	Role	Access rights to Keys and/or SSPs	Indicator
Cryptographic Services						
Symmetric encryption	Perform AES encryption	AES-CBC AES-CCM AES-CFB8 AES-CMAC AES-GMAC AES-XTS	AES key	CO	W, E	GNUTLS_FIPS140_OP_APPROVED
Symmetric decryption	Perform AES decryption	AES-CBC AES-CCM AES-CFB8 AES-CMAC AES-GMAC AES-XTS	AES key		W, E	GNUTLS_FIPS140_OP_APPROVED
Authenticated symmetric encryption	Encrypt a plaintext	AES-CCM	AES key		W, E	GNUTLS_FIPS140_OP_APPROVED
Authenticated symmetric decryption	Decrypt a ciphertext	AES-CCM	AES key		W, E	GNUTLS_FIPS140_OP_APPROVED
Key wrapping	Key wrapping (as part of the cipher suites in the TLS protocol)	AES-CCM AES-GCM	AES key		W, E	GNUTLS_FIPS140_OP_APPROVED
		AES-CBC, HMAC	AES key		W, E	
			HMAC key		W, E	
Key unwrapping	Key unwrapping (as part of the cipher suites in the TLS protocol)	AES-CCM AES-GCM	AES key		W, E	GNUTLS_FIPS140_OP_APPROVED
		AES-CBC, HMAC	AES key		W, E	
			HMAC key		W, E	
Key pair generation	Generate RSA, DH, ECDH and ECDSA key pairs	CKG DRBG Safe primes key pair generation RSA ECDSA	Module-generated Diffie-Hellman public key		G, E, R	GNUTLS_FIPS140_OP_APPROVED
			Module-generated Diffie-Hellman private key		G, E, R	
			Module-generated RSA public key		G, E, R	
			Module-generated RSA private key		G, E, R	

Service	Description	Approved Security Functions	Keys and/or SSPs	Role	Access rights to Keys and/or SSPs	Indicator
			Module-generated ECDSA public key		G, E, R	
			Module-generated ECDSA private key		G, E, R	
			Module-generated EC Diffie-Hellman private key		G, E, R	
			Module-generated EC Diffie-Hellman public key		G, E, R	
Digital signature generation	Generate RSA and ECDSA signature	SHA, RSA ECDSA	RSA private key		W, E	GNUTLS_FIPS140_OP_APPROVED
			ECDSA private key			
Digital signature verification	Verify RSA, and ECDSA signature	SHA RSA ECDSA	RSA public key		W, E	GNUTLS_FIPS140_OP_APPROVED
			ECDSA public key			
Public key verification	Verify ECDSA public key	ECDSA	ECDSA public key		W, E	GNUTLS_FIPS140_OP_APPROVED
Random number generation	Generate random bitstrings	DRBG, Non-Physical Entropy Source	Entropy input		W, E	GNUTLS_FIPS140_OP_APPROVED
			DRBG internal state: V value, key		G, E	GNUTLS_FIPS140_OP_APPROVED
			DRBG seed		E, G	
Message digest	Compute SHA hashes	SHA	None		N/A	GNUTLS_FIPS140_OP_APPROVED
Message authentication code (MAC)	Compute HMAC	HMAC	HMAC key		W, E	GNUTLS_FIPS140_OP_APPROVED
	Compute AES-based CMAC	AES-CMAC	AES key		W, E	
	Compute AES-based GMAC	AES-GMAC	AES key		W, E	
Diffie-Hellman shared secret computation	Perform shared secret computation	KAS-FFC-SSC	Diffie-Hellman public key		W, E	GNUTLS_FIPS140_OP_APPROVED
			Diffie-Hellman private key		W, E	
			Diffie-Hellman shared secret		G, R	

Service	Description	Approved Security Functions	Keys and/or SSPs	Role	Access rights to Keys and/or SSPs	Indicator
EC Diffie-Hellman shared secret computation	Perform shared secret computation	KAS-ECC-SSC	EC Diffie-Hellman public key		W, E	GNUTLS_FIPS140_OP_APPROVED
			EC Diffie-Hellman private key		W, E	
			EC Diffie-Hellman shared secret		G, R	
HKDF key derivation	Perform key derivation using HKDF (in the context of TLS 1.3)	KDA HKDF	Diffie-Hellman shared Secret		W, E	GNUTLS_FIPS140_OP_APPROVED
			EC Diffie-Hellman shared secret		W, E	
			HKDF derived key		G, R	
Password-based key derivation	Perform password-based key derivation	PBKDF	Password or passphrase		W, E	GNUTLS_FIPS140_OP_APPROVED
			PBKDF derived key		G, R	
TLS KDF key derivation	Perform key derivation using TLS 1.0/1.1, 1.2 KDF	TLS KDF v1.0/1.1 TLS KDF v1.2 RFC7627	TLS Pre-master secret		W, E	GNUTLS_FIPS140_OP_APPROVED
			TLS Master secret		W, E, G	
			TLS Derived key		G, R	
Network Protocol Services						
Transport Layer Security (TLS) network protocol	Establish TLS session	Supported cipher suites in FIPS-validated configuration (see Appendix A for the complete list of valid cipher suites)	RSA public key, RSA private key ECDSA public key, ECDSA private key	CO	W, E	GNUTLS_FIPS140_OP_APPROVED
			Diffie-Hellman public key, EC Diffie-Hellman public key		W, E, G, R	

Service	Description	Approved Security Functions	Keys and/or SSPs	Role	Access rights to Keys and/or SSPs	Indicator
			TLS pre-master secret, TLS Master secret, TLS Derived key, HKDF derived key, Diffie-Hellman private key, EC Diffie-Hellman private key		E, G	
Other FIPS-Related Services						
Show status	Show module status	N/A	None	CO	N/A	Implicit (always approved)
Zeroization	Zeroize SSPs	N/A	All SSPs		Z	Implicit (always approved)
Self-tests	Perform self-tests	AES, Diffie-Hellman, EC Diffie-Hellman, ECDSA, DRBG, HMAC, RSA, SHS, HKDF KDA, TLSv1.2 KDF (RFC7627)	None		N/A	Implicit (always approved)
Show module name and version	Show module name and version	N/A	None		N/A	Implicit (always approved)

Table 10 - Approved Services

Table 11 lists the non-approved services. The details of the non-approved cryptographic algorithms available in non-Approved mode can be found in Table 7.

Service	Description	Algorithms Accessed	Role
Symmetric encryption	Compute the cipher for encryption	AES GCM when not used in the context of the TLS protocol. Blowfish Camellia CAST ChaCha20 DES GOST RC2, RC4 Salsa20 SEED Serpent Triple-DES Twofish	CO
Symmetric decryption	Compute the cipher for decryption	AES GCM when not used in the context of the TLS protocol. Blowfish Camellia CAST ChaCha20 DES GOST RC2, RC4 Salsa20 SEED Serpent Triple-DES Twofish	
Key pair generation	Generate RSA, DSA, and ECDSA key pairs	DSA ECDSA with curves not listed in Table 5 RSA with keys smaller than 2048 bits or greater than 4096 bits	
Digital signature generation	Sign RSA, DSA, and ECDSA signatures	DSA ECDSA with curves not listed in Table 5 RSA with keys smaller than 2048 bits or greater than 4096 bits SHA-1	
Digital signature verification	Verify RSA, DSA, and ECDSA signatures	DSA ECDSA with curves not listed in Table 5 RSA with keys smaller than 1024 bits or greater than 4096 bits.	
Domain parameter generation	Generate domain parameter	DSA	

Service	Description	Algorithms Accessed	Role
Message digest	Compute message digest	GOST MD2, MD4, MD5 RMD160 STREEBOG	
Message authentication code (MAC)	Compute message authentication code	CMAC with Triple-DES GMAC HMAC with keys smaller than 112-bit HMAC with GOST MD2, MD4, MD5 RMD160 STREEBOG UMAC	
Key encapsulation	Perform RSA key encapsulation	RSA encryption and decryption with any key sizes	
Key un-encapsulation	Perform RSA key un-encapsulation	RSA encryption and decryption with any key sizes	
Diffie-Hellman shared secret computation	Shared secret computation using DH	Diffie-Hellman with keys generated with domain parameters other than safe primes	
EC Diffie-Hellman shared secret computation	Shared secret computation using ECDH	EC Diffie-Hellman with curves not listed in Table 5	
Key agreement	Perform key agreement	Diffie-Hellman with keys generated with domain parameters other than safe primes EC Diffie-Hellman with curves not listed in Table 5 SRP	
Password-based key derivation	Perform key derivation using PBKDF	PBKDF with non-approved message digest algorithms	
Public key verification	Verify ECDSA public key	ECDSA with curves not listed in Table 5.	
Transport Layer Security (TLS) Network Protocol	Establish non-supported TLS channel	Non-supported cipher suites (see Appendix A for the complete list of valid cipher suites)	

Table 11 - Non-Approved Services

5 Software/Firmware security

5.1 Integrity Techniques

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

5.2 On-Demand Integrity Test

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

5.3 Executable Code

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

6 Operational Environment

6.1 Applicability

This module operates in a modifiable operational environment per the FIPS 140-3 level 1 specifications. The SUSE Linux Enterprise Server operating system is used as the basis of other products. Compliance is maintained for SUSE products whenever the binary is found unchanged per the vendor affirmation from SUSE based on the allowance FIPS 140-3 management manual section 7.9.1 bullet 1 a i).

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

6.2 Policy

The module does not support concurrent operators.

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

6.3 Requirements

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

7 Physical Security

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

8 Non-invasive Security

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

9 Sensitive Security Parameter Management

Table 12 summarizes the SSPs that are used by the cryptographic services implemented in the module.

Key / SSP Name / Type	Strength	Security Function and Cert. Number[3]	Generation	Import/Export	Establishment	Storage	Zeroization	Use & related keys
AES key	AES-XTS: 128, 256 Other modes: 128, 192, 256	AES-CBC, AES-CCM, AES-CFB8, AES-CMAC, AES-GCM, AES-GMAC, AES-XTS A2984, A2985, A2986, A2987, A2989, A2990, A2992, A2993, A2995, A2996, A2997, A3004, A3007	N/A	MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None	N/A	RAM	gnutls_cipher_deinit() gnutls_aead_cipher_deinit()	**Use:** Symmetric encryption; Symmetric decryption; Authenticated symmetric encryption; Authenticated symmetric decryption; Message authentication code (MAC); Key wrapping; Key unwrapping; **Related SSPs:** N/A
HMAC key	112-256	HMAC A2987, A2992, A2998, A3007	N/A	MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None	N/A	RAM	gnutls_hmac_deinit()	**Use:** Message Authentication Code (MAC); Key wrapping; Key unwrapping; **Related SSPs:** N/A
Module-generated RSA public key	112 to 256	RSA CTR_DRBG A2992	Generated using the FIPS 186-4 key generation method; the random value used in key generation is obtained from the SP800-90Arev1 DRBG.	MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None	N/A	RAM	gnutls_privkey_deinit() gnutls_x509_privkey_deinit() gnutls_rsa_params_deinit()	**Use:** Key pair generation **Related SSPs:** DRBG internal state: V value, key; Module-generated RSA private key
Module-generated RSA private key	112 to 256	RSA CTR_DRBG A2992	Generated using the FIPS 186-4 key generation method; the random value used in key generation is obtained from	MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format.	N/A	RAM	gnutls_privkey_deinit() gnutls_x509_privkey_deinit() gnutls_rsa_params_deinit()	**Use:** Key pair generation **Related SSPs:** DRBG internal state: V value, key; Module-generated RSA public key

[3] see Table 5 for the certificate number of each algorithm listed in this column.

Key / SSP Name / Type	Strength	Security Function and Cert. Number[3]	Generation	Import/Export	Establishment	Storage	Zeroization	Use & related keys
			the SP800-90Arev1 DRBG.	**Import:** None				
RSA private key	112-256	RSA A2992	N/A	MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None	N/A	RAM	gnutls_privkey_deinit() gnutls_x509_privkey_deinit() gnutls_rsa_params_deinit()	**Use:** Digital signature generation; Transport Layer Security (TLS) network protocol **Related SSPs:** RSA public key
RSA public key	112-256	RSA A2992	N/A	MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None	N/A	RAM	gnutls_privkey_deinit() gnutls_x509_privkey_deinit() gnutls_rsa_params_deinit()	**Use:** Digital signature verification; Transport Layer Security (TLS) network protocol **Related SSPs:** RSA private key
Module-generated ECDSA private key	128, 192, 256	ECDSA CTR_DRBG A2992	Generated using the FIPS 186-4 key generation method; the random value used in key generation is obtained from the SP800-90Arev1 DRBG.	MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None	N/A	RAM	gnutls_pk_params_clear()	**Use:** Key pair generation **Related SSPs:** DRBG internal state: V value, key; Module-generated ECDSA public key
Module-generated ECDSA public key	128, 192, 256	ECDSA CTR_DRBG A2992	Generated using the FIPS 186-4 key generation method; the random value used in key generation is obtained from the SP800-90Arev1 DRBG	MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None	N/A	RAM	gnutls_pk_params_clear()	**Use:** Key pair generation **Related SSPs:** DRBG internal state: V value, key; Module-generated ECDSA private key
ECDSA public key	128, 192, 256	ECDSA A2992	N/A	MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None	N/A	RAM	gnutls_pk_params_clear()	**Use:** Digital signature verification; Public key verification; Transport Layer Security (TLS) network protocol **Related SSPs:** ECDSA private key

Key / SSP Name / Type	Strength	Security Function and Cert. Number[3]	Generation	Import/Export	Establishment	Storage	Zeroization	Use & related keys
ECDSA private key	128, 192, 256	ECDSA A2992	N/A	MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None	N/A	RAM	gnutls_pk_params_clear()	**Use:** Digital signature generation; Transport Layer Security (TLS) network protocol; Public key verification; **Related SSPs:** ECDSA public key
Module-generated Diffie-Hellman public key	112 to 200	KAS-FFC-SSC CTR_DRBG A2992	Generated using the SP 800-56Arev3 Safe Primes key generation method; random values are obtained from the SP800-90Arev1 DRBG.	MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None	N/A	RAM	gnutls_dh_params_deinit() gnutls_pk_params_clear()	**Use:** Key pair generation; Transport Layer Security (TLS) network protocol **Related SSPs:** Module-generated Diffie-Hellman private key; DRBG internal state: V value, key; TLS pre-master secret
Module-generated Diffie-Hellman private key	112 to 200	KAS-FFC-SSC CTR_DRBG A2992	Generated using the SP 800-56Arev3 Safe Primes key generation method; random values are obtained from the SP800-90Arev1 DRBG.	MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None	N/A	RAM	gnutls_dh_params_deinit() gnutls_pk_params_clear()	**Use:** Key pair generation; Transport Layer Security (TLS) network protocol **Related SSPs:** Module-generated Diffie-Hellman public key; DRBG internal state: V value, key; TLS pre-master secret
Diffie-Hellman public key	112 to 200	KAS-FFC-SSC A2992		MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None	N/A	RAM	gnutls_dh_params_deinit() gnutls_pk_params_clear()	**Use:** Diffie-Hellman shared secret computation; Transport Layer Security (TLS) network protocol **Related keys:** Diffie-Hellman private key; Diffie-Hellman shared secret
Diffie-Hellman private key	112 to 200	KAS-FFC-SSC A2992		MD/EE	N/A	RAM	gnutls_dh_params_deinit() gnutls_pk_params_clear()	**Use:** Diffie-Hellman shared secret computation;

Key / SSP Name / Type	Strength	Security Function and Cert. Number[3]	Generation	Import/Export	Establishment	Storage	Zeroization	Use & related keys
				Import: CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None				Transport Layer Security (TLS) network protocol **Related keys:** Diffie-Hellman public key; Diffie-Hellman shared secret
Module-generated EC Diffie-Hellman private key	128, 192, 256	KAS-ECC-SSC CTR_DRBG A2992	Generated internally by the module using the ECDSA key generation method compliant with [FIPS186-4] and [SP800-56Arev3]; the random value used in key generation is obtained from the SP800-90Arev1 DRBG	MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None	N/A	RAM	gnutls_pk_params_clear()	**Use:** Key pair generation; Transport Layer Security (TLS) network protocol **Related keys:** Module-generated EC Diffie-Hellman public key; DRBG internal state: V value, key; TLS pre-master secret
Module-generated EC Diffie-Hellman public key	128, 192, 256	KAS-ECC-SSC CTR_DRBG A2992	Generated internally by the module using the ECDSA key generation method compliant with [FIPS186-4] and [SP800-56Arev3]; the random value used in key generation is obtained from the SP800-90Arev1 DRBG	MD/EE **Export:** CM to TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None	N/A	RAM	gnutls_pk_params_clear()	**Use:** Key pair generation; Transport Layer Security (TLS) network protocol **Related keys:** Module-generated EC Diffie-Hellman private key; DRBG internal state: V value, key; TLS pre-master secret
EC Diffie-Hellman private key	128, 192, 256	KAS-ECC-SSC A2992	N/A	MD/EE **Import:** CM from TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None	N/A	RAM	gnutls_pk_params_clear()	**Use:** EC Diffie-Hellman shared secret computation; **Related keys:** EC Diffie-Hellman shared secret; EC Diffie-Hellman public key
EC Diffie-Hellman public key	128, 192, 256	KAS-ECC-SSC A2992	N/A	MD/EE **Import:** CM from TOEPP Path.	N/A	RAM	gnutls_pk_params_clear()	**Use:** EC Diffie-Hellman shared secret computation; **Related keys:** EC

Key / SSP Name / Type	Strength	Security Function and Cert. Number[3]	Generation	Import/Export	Establishment	Storage	Zeroization	Use & related keys
				Passed to the module via API parameters in plaintext (P) format. **Export:** None				Diffie-Hellman private key; EC Diffie-Hellman shared secret
Diffie-Hellman shared secret	112 to 200	KAS-FFC-SSC A2992	N/A	MD/EE **Import:** CM to/from TOEPP Path. Passed to/from the module via API parameters in plaintext (P) format **Export:** CM from TOEPP Path. Passed from the module via API parameters in plaintext (P) format.	Generated during the Diffie-Hellman key agreement and shared secret computation per SP800-56Arev3.	RAM	zeroize_key()	**Use:** Diffie-Hellman shared secret computation; HKDF key derivation **Related keys:** Diffie-Hellman public key, Diffie-Hellman private key;
EC Diffie-Hellman shared secret	112 to 256	KAS-ECC-SSC A2992	N/A	MD/EE **Import:** CM to/from TOEPP Path. Passed to/from the module via API parameters in plaintext (P) format. **Export:** CM from TOEPP Path. Passed from the module via API parameters in plaintext (P) format.	Generated during the EC Diffie-Hellman key agreement and shared secret computation per SP800-56Arev3.	RAM	zeroize key()	**Use:** EC Diffie-Hellman shared secret computation; HKDF key derivation **Related keys:** EC Diffie-Hellman public key; EC Diffie-Hellman private key;
PBKDF password or passphrase	Password strength 10^{20} - 10^{128}	PBKDF A2992	N/A (key material is entered via API parameters)	MD/EE **Import:** CM to TOEPP Path. Passed to the module via API parameters in plaintext (P) format. **Export:** None	N/A	RAM	Internal PBKDF state is zeroized automatically when function returns.	**Use:** Password-based key derivation **Related keys:** PBKDF derived key
PBKDF derived key	112-256 bits	PBKDF A2992	Derived during the PBKDF	MD/EE **Export:** CM from TOEPP Path. Passed from the module via API parameters in plaintext (P) format.	N/A	RAM	zeroize_key()	**Use:** Password-based key derivation **Related keys:** PBKDF password or passphrase

Key / SSP Name / Type	Strength	Security Function and Cert. Number[3]	Generation	Import/Export	Establishment	Storage	Zeroization	Use & related keys
				Import: None				
Entropy input IG D.L compliant	192 to 384 bits	DRBG A2992 ESV E28, E29	N/A	**Import:** None **Export:** None it remains within the cryptographic boundary.	N/A	RAM	gnutls_global_deinit()	**Use:** Random number generation **Related keys:** DRBG seed
DRBG seed IG D.L compliant	192 to 384 bits	CTR_DRBG A2992 ESV E28, E29	Generated from the entropy input as defined in SP800-90Arev1	**Import:** None **Export:** None it remains within the cryptographic boundary.	N/A	RAM	gnutls_global_deinit()	**Use:** Random number generation **Related SSPs:** Entropy input; DRBG internal state: V value, key
DRBG internal state: V value, key IG D.L compliant	128 to 256 bits	CTR_DRBG A2992	Generated from the DRBG seed as defined in SP800-90Arev1	**Import:** None **Export:** None	N/A	RAM	gnutls_global_deinit()	**Use:** Random number generation **Related keys:** DRBG seed, Module-generated ECDSA public key, Module-generated ECDSA private key, Module-generated RSA public key, Module-generated RSA private key, Module-generated Diffie-Hellman public key, Module-generated Diffie-Hellman private key, Module-generated EC Diffie-Hellman public key, Module-generated EC Diffie-Hellman private key
TLS pre-master secret	DH 112 to 200 ECDH 112 to 256 bits	KDF TLS, TLS v1.2 KDF RFC7627 A2992	N/A	MD/EE **Export:** None **Import:** CM to TOEPP Path. Passed to the module via API parameters in plaintext (P) format.	Generated during the EC Diffie-Hellman / Diffie-Hellman key agreement and shared secret computat	RAM	gnutls_deinit()	**Use:** Transport Layer Security (TLS) network protocol; TLS key derivation **Related keys:** TLS master secret

Key / SSP Name / Type	Strength	Security Function and Cert. Number[3]	Generation	Import/Export	Establishment	Storage	Zeroization	Use & related keys
					ion per SP800-56Arev3.			
TLS master secret	112 to 256 bits	KDF TLS, TLS v1.2 KDF RFC7627 A2992	Derived from TLS pre-master secret using TLS KDF per SP800-135rev1 (TLSv1.0/1.1) TLS v1.2 KDF RFC7627	MD/EE **Export:** None **Import:** None	N/A	RAM	gnutls_deinit()	**Use:** Transport Layer Security (TLS) network protocol; TLS key derivation **Related keys:** TLS pre-master secret, TLS Derived key
TLS Derived key	112 to 256 bits	KDF TLS, TLS v1.2 KDF RFC7627 A2992	Derived from TLS master secret during the TLS KDF per SP800-135rev1 (TLSv1.0/1.1) TLS v1.2 KDF RFC7627	MD/EE **Export:** CM from TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None	N/A	RAM	gnutls_deinit()	**Use:** Transport Layer Security (TLS) network protocol; TLS key derivation **Related keys:** TLS pre-master secret; TLS master secret
HKDF derived key	112 to 256 bits	KDA HKDF A2991	Derived (as part of TLSv1.3) with KDA HKDF	MD/EE **Export:** CM from TOEPP Path. Passed from the module via API parameters in plaintext (P) format. **Import:** None	N/A	RAM	gnutls_deinit()	**Use:** Transport Layer Security (TLS) network protocol; HKDF key derivation **Related keys:** EC Diffie-Hellman shared secret; Diffie-Hellman shared secret

Table 12 - SSPs

9.1 Random Number Generation

The module employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90Arev1] for the creation of seeds for asymmetric keys, random numbers for security functions (e.g. ECDSA signature generation), 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 an SP800-90B-compliant entropy source specified in Table 13. This entropy source is located within the physical perimeter, but outside of the cryptographic boundary of the module. The module obtains 384 bits to seed the DRBG, and 256 bits to reseed it, sufficient to provide a DRBG with 256 bits of security strength.

Entropy Sources	Minimum number of bits of entropy	Details

Non-Physical Entropy Source ESV E28, E29	256 bits of entropy in the 256-bit output	Userspace Standalone CPU Time Jitter RNG version 3.4.0 entropy source (using SHA-3 as the vetted conditioning component) is located within the physical perimeter of the operational environment but outside the module cryptographic boundary.

Table 13 - Non-Deterministic Random Number Generation Specification

9.2 SSP Generation

In accordance with FIPS 140-3 IG D.H, the cryptographic module performs Cryptographic Key Generation (CKG) for asymmetric keys according to section 4, 5.1 and 5.2 of [SP800-133rev2] by obtaining a random bit string directly from an approved [SP800-90Arev1] DRBG and that can support the required security strength requested by the caller (without any V, as described in Additional Comments 2 of IG D.H).

- For generating RSA and ECDSA keys, the module implements asymmetric cryptographic key generation (CKG) services compliant with [FIPS186-4].
- The public and private keys used in the EC Diffie-Hellman key agreement schemes are generated internally by the module using the ECDSA key generation method compliant with [FIPS186-4] and [SP800-56Arev3].
- The public and private keys used in the Diffie-Hellman key agreement scheme are also compliant with [SP800-56Arev3]. The module generates keys using safe primes defined in RFC7919 and RFC3526, as described in the next section.

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

- RSA [FIPS186-4] B.3.2 Random Provable Primes - 2048, 3072, 4096-bit keys with 112-149 bits of key strength
- ECDH/ECDSA [FIPS186-4] B.4.2 Testing Candidates - P-256, P-384, P-521 elliptic curves with 128-256 bits of key strength
- Safe Primes Key Generation [SP800-56Arev3] - 2048, 3072, 4096, 6144, 8192-bit keys with 112-200 bits of key strength.

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

- KDF for the TLS protocol, used as pseudo-random functions (PRF) for TLSv1.0/1.1 and TLSv1.2 (RFC7627).

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

- HKDF for the TLS protocol TLSv1.3.

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

9.3 SSP establishment

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

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

- TLS (RFC7919)
  - ffdhe2048 (ID = 256)
  - ffdhe3072 (ID = 257)
  - ffdhe4096 (ID = 258)
  - ffdhe6144 (ID = 259)
  - ffdhe8192 (ID = 260)

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

- IKEv2 (RFC3526)
  - MODP-2048 (ID=14)
  - MODP-3072 (ID=15)
  - MODP-4096 (ID=16)
  - MODP-6144 (ID=17)
  - MODP-8192 (ID=18)

The module also provides the following key transport mechanisms:

- Key wrapping using AES-CCM, AES-GCM, and AES-CBC with HMAC, used in the context of the TLS protocol cipher suites (in compliance with IG D.G) with 128-bit or 256-bit keys, providing 128, 256 bits of key strength.

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

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

SP 800-56Ar3 Assurances

To comply with the assurances found in Section 5.6.2 of SP 800-56Ar3, the operator must use the module together with an application that implements the TLS protocol. Additionally, the module’s approved “Key pair generation” service must be used to generate ephemeral Diffie- Hellman or EC Diffie-Hellman key pairs, or the key pairs must be obtained from another FIPS-validated module.

As part of this service, the module will internally perform the full public key validation of the generated public key. The module’s shared secret computation service will internally perform the full public key validation of the peer public key, complying with Sections 5.6.2.2.1 and 5.6.2.2.2 of SP 800-56Ar3.

9.4 SSP Entry and Output

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

9.5 SSP Storage

All SSPs not generated by the module are provided by the calling application.

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

9.6 SSP Zeroization

The memory occupied by SSPs is allocated by regular memory allocation operating system calls. The application that is acting as the CO is responsible for calling the appropriate zeroization functions provided in the module's API and listed in Table 12. Calling the gnutls_global_deinit() will zeroize the SSPs stored in the TLS protocol internal state and also invoke the corresponding API functions listed in Table 12 to zeroize SSPs. The zeroization functions overwrite the memory occupied by SSPs with “zeros” and deallocate the memory with the regular memory deallocation operating system call. The completion of a zeroization routine(s) will indicate that a zeroization procedure succeeded.

10 Self-tests

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

10.1 Pre-Operational Tests

The module performs the integrity test using HMAC-SHA2-256. The details of integrity test are provided in 5.1.

10.2 Conditional Tests

10.2.1 Cryptographic algorithm tests

Table 14 specifies all the CASTs. The CASTs are performed in the form of the Known Answer Tests (KATs) and are run prior to performing the integrity test. A KAT includes the comparison of a calculated output with an expected known answer, hard coded as part of the test vectors used in the test. If the values do not match, the KAT fails.

Algorithm	Test
AES	KAT AES CBC mode with 128, 256-bit keys, 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); KAT AES CFB8 mode with 256-bit keys, encryption and decryption (separately tested); KAT AES CMAC mode with 256-bit keys, encryption and decryption (separately tested);
Diffie-Hellman	Primitive “Z” Computation KAT with 3072-bit key using ffdhe3072 safe-prime.
DRBG	KAT CTR_DRBG with AES with 256-bit keys without DF, without PR Health tests according to section 11.3 of [SP800-90Arev1]
EC Diffie-Hellman	Primitive “Z” Computation KAT with P-256 curve
ECDSA	KAT ECDSA with P-256 using SHA-256, P-384 using SHA-384, and P-521 using SHA-512, signature generation and verification (separately tested)
HKDF KDA	KAT with HMAC-SHA2-256
HMAC	KAT HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC- SHA2-512

Algorithm	Test
PBKDF2 KDF	KAT with HMAC-SHA2-256
RSA	KAT RSA with 2048-bit key using SHA2-256, signature generation and verification (separately tested);
SHA-3	KAT SHA3-224, SHA3-256, SHA3-384, SHA3-512
TLSv1.2 KDF (RFC7627)	KAT with SHA2-256

Table 14 - Conditional Cryptographic Algorithms Self-Tests

10.2.2 Pairwise Consistency Test

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

Algorithm	Test
ECDSA key generation	PCT using SHA2-256, signature generation and verification.
RSA key generation	PCT using SHA2-256, signature generation and verification.
Diffie-Hellman key generation	PCT according to section 5.6.2.1.4 of [SP800-56Arev3]
EC Diffie-Hellman key generation	Covered by ECDSA PCT as allowed by IG 10.3.A additional comment 1

Table 15 - Pairwise Consistency Test

10.2.3 Periodic/On-Demand Self-Test

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

10.3 Error States

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

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

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

Table 16 - Error States

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

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

11 Life-cycle assurance

The following sections describe the Delivery and Operation and Crypto Officer Guidance of the module.

11.1 Delivery and Operation

11.1.1 Module Installation

The Crypto Officer can install the RPM packages containing the module as listed in Table 18 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.

11.1.2 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.

11.1.3 Module Installation for Vendor Affirmed Platforms

Table 17 includes the information on module installation process for the vendor affirmed platforms that are listed in Table 4.

Product	Link
SUSE Linux Enterprise Micro 5.3	https://documentation.suse.com/sle-micro/5.3/single-html/SLE-Micro-security/#sec-fips-slemicro-install
SUSE Linux Enterprise Server for SAP 15SP4	https://documentation.suse.com/sles/15-SP4/html/SLES-all/book-security.html
SUSE Linux Enterprise Base Container Image 15SP4	https://documentation.suse.com/smart/linux/html/concept-bci/index.html
SUSE Linux Enterprise Desktop 15SP4	https://documentation.suse.com/sled/15-SP4/html/SLED-all/book-security.html
SUSE Linux Enterprise Real Time 15SP4	https://documentation.suse.com/sle-rt/15-SP4/

Table 17 - Installation for Vendor Affirmed Platforms

Note: Per section 7.9 in the FIPS 140-3 Management Manual [FIPS140-3_MM], 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.

11.1.4 End of Life Procedure

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

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

11.2 Crypto Officer Guidance

The binaries of the module are contained in the RPM packages for delivery. The Crypto Officer shall follow section 11.1.1 and 11.1.2 to configure the operational environment and install the module to be operated as a FIPS 140-3 validated module.

Table 16 lists the RPM packages that contain the FIPS validated module. The "Show module name and version" service returns the value “GnuTLS version 3.7.3-150400.4.35.1”, which matches the version included in the RPM package filenames, and map to version 1.1 of the cryptogtaphic module.

Processor Architecture	RPM Packages
Intel 64-bit	libgnutls30-3.7.3-150400.4.35.1.x86_64.rpm libnettle8-3.7.3-150400.2.21.x86_64.rpm libhogweed6-3.7.3-150400.2.21.x86_64.rpm libgmp10-6.1.2-4.9.1.x86_64.rpm
AMD 64-bit	libgnutls30-3.7.3-150400.4.35.1.x86_64.rpm

Processor Architecture	RPM Packages
	libnettle8-3.7.3-150400.2.21.x86_64.rpm libhogweed6-3.7.3-150400.2.21.x86_64.rpm libgmp10-6.1.2-4.9.1.x86_64.rpm
IBM z15	libgnutls30-3.7.3-150400.4.35.1.s390x.rpm libnettle8-3.7.3-150400.2.21.s390x.rpm libhogweed6-3.7.3-150400.2.21.s390x.rpm libgmp10-6.1.2-4.9.1.s390x.rpm
ARMv8 64-bit	libgnutls30-3.7.3-150400.4.35.1.aarch64.rpm libnettle8-3.7.3-150400.2.21.aarch64.rpm libhogweed6-3.7.3-150400.2.21.aarch64.rpm libgmp10-6.1.2-4.9.1.aarch64.rpm
IBM Power10 64-bit	libgnutls30-3.7.3-150400.4.35.1.ppc64le.rpm libnettle8-3.7.3-150400.2.21.ppc64le.rpm libhogweed6-3.7.3-150400.2.21.ppc64le.rpm libgmp10-6.1.2-4.9.1.ppc64le.rpm

Table 18 - RPM packages

11.2.1 TLS

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

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

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

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

11.2.2 AES XTS

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

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

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

11.2.3 AES GCM IV

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

- For TLS v1.2, IV generation is in compliance with scenario 1.a of IG C.H and [RFC5288]. The module supports acceptable AES-GCM ciphersuites from section 3.3.1 of [SP800-52rev2].
- For TLS v1.3, IV generation is in compliance with scenario 5 of IG C.H and [RFC8446]. The module supports acceptable AES-GCM ciphersuites from section 3.3.1 of [SP800-52rev2].

The IV generated in both scenarios is only used within the context of the TLS protocol implementation. The nonce_explicit part of the IV does not exhaust the maximum number of possible values for a given session key. The design of the TLS protocol in this module implicitly ensures that the nonce_explicit, or counter portion of the IV will not exhaust all of its possible values.

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

11.2.4 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 the approved mode of operation:

- gnutls_crypto_register_cipher
- gnutls_crypto_register_aead_cipher
- gnutls_crypto_register_mac
- gnutls_crypto_register_digest
- gnutls_privkey_import_ext4

11.2.5 Key derivation using SP800-132 PBKDF

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

In accordance with [SP800-132], the following requirements shall be met.

- Derived keys shall only be used in storage applications. The Master Key (MK) shall not be used for other purposes. The length of the MK or DPK shall be of 112 bits or more (this is verified by the module to determine the service is approved).
- A portion of the salt, with a length of at least 128 bits (this is verified by the module to determine the service is approved), shall be generated randomly using the SP800-90Arev1 DRBG,
- The iteration count shall be selected as large as possible, as long as the time required to generate the key using the entered password is acceptable for the users. The minimum value shall be 1000 (this is verified by the module to determine the service is approved).
- Passwords or passphrases, used as an input for the PBKDF, shall not be used as cryptographic keys.

- The length of the password or passphrase shall be of at least 20 characters (this is verified by the module to determine the service is approved), and shall consist of lower-case, upper-case and numeric characters. The probability of guessing the value is estimated to be 10^{-20} (assuming all digits).

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

12 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 version 1.0, 1.1, 1.2 and 1.3, compliant with section 3.3.1 of [SP800-52rev2]. Each cipher suite defines the key exchange algorithm, the bulk encryption algorithm (including the symmetric key size) and the MAC algorithm.

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

Cipher Suite	ID	Reference
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256	{ 0xC0, 0x25 }	RFC5289
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384	{ 0xC0, 0x26 }	RFC5289
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	{ 0xC0, 0x27 }	RFC5289
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	{ 0xC0, 0x28 }	RFC5289
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	{ 0xC0, 0x29 }	RFC5289
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384	{ 0xC0, 0x2A }	RFC5289
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	{ 0xC0, 0x2B }	RFC5289
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	{ 0xC0, 0x2C }	RFC5289
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	{ 0xC0, 0x2D }	RFC5289
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384	{ 0xC0, 0x2E }	RFC5289
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	{ 0xC0, 0x2F }	RFC5289
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	{ 0xC0, 0x30 }	RFC5289
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	{ 0xC0, 0x31 }	RFC5289
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	{ 0xC0, 0x32 }	RFC5289
TLS_DHE_RSA_WITH_AES_128_CCM	{ 0xC0, 0x9E }	RFC6655
TLS_DHE_RSA_WITH_AES_256_CCM	{ 0xC0, 0x9F }	RFC6655
TLS_DHE_RSA_WITH_AES_128_CCM_8	{ 0xC0, 0xA2 }	RFC6655
TLS_DHE_RSA_WITH_AES_256_CCM_8	{ 0xC0, 0xA3 }	RFC6655
TLS_AES_128_GCM_SHA256	{ 0x13, 0x01 }	RFC8446
TLS_AES_256_GCM_SHA384	{ 0x13, 0x02 }	RFC8446
TLS_AES_128_CCM_SHA256	{ 0x13, 0x04 }	RFC8446
TLS_AES_128_CCM_8_SHA256	{ 0x13, 0x05 }	RFC8446

Table 19 - TLS Cipher Suites

Appendix B. Glossary and Abbreviations

AES	Advanced Encryption Standard
AES-NI	Advanced Encryption Standard New Instructions
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CCM	Counter with Cipher Block Chaining-Message Authentication Code
CFB	Cipher Feedback
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CPACF	Central Processor Assist for Cryptographic Function
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DF	Derivation Function
DSA	Digital Signature Algorithm
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standards Publication
FSM	Finite State Model
GCM	Galois Counter Mode
HMAC	Hash Message Authentication Code
KAS	Key Agreement Schema
KAT	Known Answer Test
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
OFB	Output Feedback
O/S	Operating System
PAA	Processor Algorithm Acceleration
PAI	Processor Algorithm Implementation
PR	Prediction Resistance
PSS	Probabilistic Signature Scheme
RNG	Random Number Generator
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm

SHS	Secure Hash Standard
SSH	Secure Shell
SSP	Sensitive Security Parameter
TDES	Triple-DES
XTS	XEX-based Tweaked-codebook mode with cipher text Stealing

Appendix C. References

FIPS140-3	**FIPS PUB 140-3 - Security Requirements For Cryptographic Modules** March 2019 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf
FIPS140-3_IG	**Implementation Guidance for FIPS PUB 140-3 and the Cryptographic Module Validation Program** March 2024 https://csrc.nist.gov/csrc/media/Projects/cryptographic-module-validation-program/documents/fips%20140-3/FIPS%20140-3%20IG.pdf
FIPS140-3_MM	**FIPS 140-3 Cryptographic Module Validation Program - Management Manual** April 2024 https://csrc.nist.gov/csrc/media/Projects/cryptographic-module-validation-program/documents/fips%20140-3/FIPS-140-3-CMVP%20Management%20Manual.pdf
FIPS180-4	**Secure Hash Standard (SHS)** August 2015 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
FIPS186-4	**Digital Signature Standard (DSS)** July 2013 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS197	**Advanced Encryption Standard** November 2001 https://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
FIPS198-1	**The Keyed Hash Message Authentication Code (HMAC)** July 2008 https://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
FIPS202	**SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions** August 2015 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
PKCS#1	**Public Key Cryptography Standards (PKCS) #1: RSA Cryptography** Specifications Version 2.1 February 2003 https://www.ietf.org/rfc/rfc3447.txt
RFC3394	**Advanced Encryption Standard (AES) Key Wrap Algorithm** September 2002 https://www.ietf.org/rfc/rfc3394.txt

SP800-38A **NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques**
December 2001
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**
May 2005
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**
May 2004
https://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**
November 2007
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**
January 2010
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38e.pdfhttps://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf

SP800-38F **NIST Special Publication 800-38F - Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping**
December 2012
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf

SP800-38G **NIST Special Publication 800-38G - Recommendation for Block Cipher Modes of Operation: Methods for Format - Preserving Encryption**
March 2016
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38G.pdf

SP800-52rev2 **NIST Special Publication 800-52 Revision 2 - Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations**
August 2019
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf

SP800-56Arev3	**NIST Special Publication 800-56A Revision 3 - Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography** April 2018 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf
SP800-56Crev2	**NIST Special Publication 800-56C Revision 2 - Recommendation for Key Derivation through Extraction-then-Expansion** August 2020 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr2.pdf
SP800-57rev5	**NIST Special Publication 800-57 Part 1 Revision 5 - Recommendation for Key Management Part 1: General** May 2020 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf
SP800-90Arev1	**NIST Special Publication 800-90A Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** June 2015 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
SP800-90B	**NIST Special Publication 800-90B - Recommendation for the Entropy Sources Used for Random Bit Generation** January 2018 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf
SP800-108rev1	**NIST Special Publication 800-108 Revision 1 - Recommendation for Key Derivation Using Pseudorandom Functions (Revised)** August 2022 https://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf
SP800-131Arev2	**NIST Special Publication 800-131 Revision 2 - Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths** March 2019 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf
SP800-132	**NIST Special Publication 800-132 - Recommendation for Password-Based Key Derivation - Part 1: Storage Applications** December 2010 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf
SP800-133rev2	**NIST Special Publication 800-133 Revision 2 - Recommendation for Cryptographic Key Generation** June 2020 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-133r2.pdf

SP800-135rev1 **NIST Special Publication 800-135 Revision 1 - Recommendation for Existing Application-Specific Key Derivation Functions**
December 2011
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf

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

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

RFC5288 **AES Galois Counter Mode (GCM) Cipher Suites for TLS**
August 2008
https://www.rfc-editor.org/rfc/rfc5288.txt

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