SUSE Linux Enterprise Server Kernel Crypto API Cryptographic Module version 3.0

FIPS 140-2 Non-Proprietary Security Policy

Doc version 3.0.2

Last update: 2020-11-08

Prepared by:
atsec information security corporation
9130 Jollyville Road, Suite 260
Austin, TX 78759
www.atsec.com

Table of contents

1 Introduction

1.1 Purpose

This document is the non-proprietary security policy for the SUSE Linux Enterprise Server Kernel Crypto API Cryptographic Module version 3.0. 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.

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 https://csrc.nist.gov/.

Throughout the document, “the Kernel Crypto API module” and “the module” are also used to refer to the SUSE Linux Enterprise Server Kernel Crypto API Cryptographic Module version 3.0.

1.2 External Resources / References

The SUSE website (www.suse.com) contains information about SUSE Linux Enterprise Server.

The Cryptographic Module Validation Program website (https://csrc.nist.gov/groups/STM/cmvp/) contains links to the FIPS 140-2 certificate and SUSE contact information.

Appendix A contains the glossary and abbreviations and Appendix B contains the references.

2 Cryptographic Module Specification

2.1 Module Overview

The SUSE Linux Enterprise Server Kernel Crypto API Cryptographic Module is a software cryptographic module that provides general-purpose cryptographic services. 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:

Description	Component
Static kernel binary	/boot/vmlinuz-4.12.14-150.47-default
Integrity check HMAC file for Linux kernel static binary	/boot/.vmlinuz-4.12.14-150.47-default.hmac
Cryptographic kernel object files	/lib/modules/4.12.14-150.47-default/kernel/crypto/*.ko /lib/modules/4.12.14-150.47-default/kernel/arch/x86/crypto/*.ko
Integrity test utility	/usr/lib64/libkcapi/fipscheck
Integrity check HMAC file for integrity test utility	/usr/lib64/libkcapi/.fipscheck.hmac

Table 2: Cryptographic Module Components

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

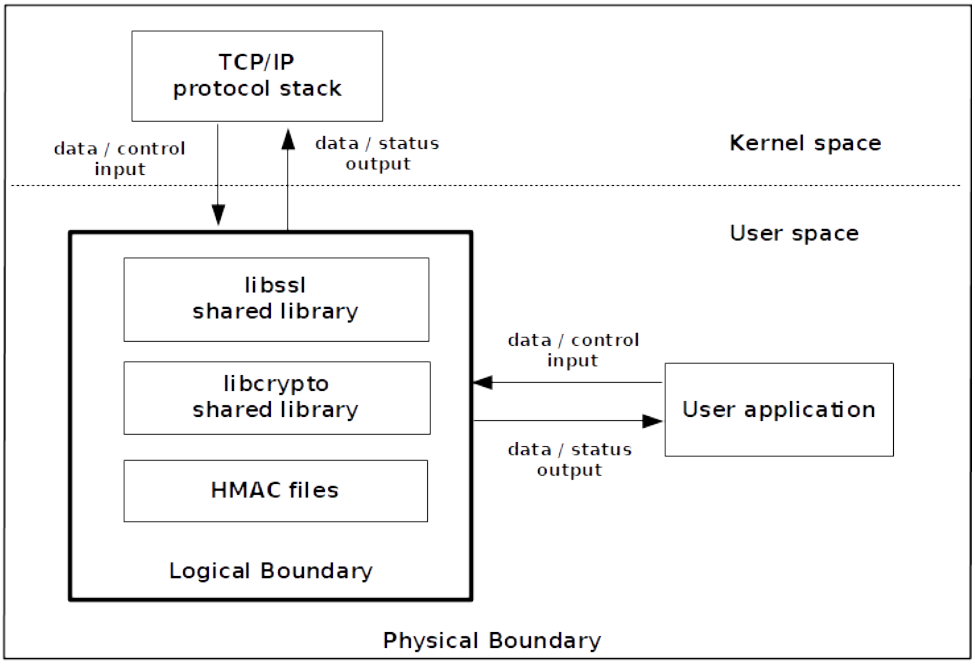

Figure 1: Software Block Diagram

The module is aimed to run on a general purpose computer (GPC). Table 3 shows the platforms 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 with and without AES-NI (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.

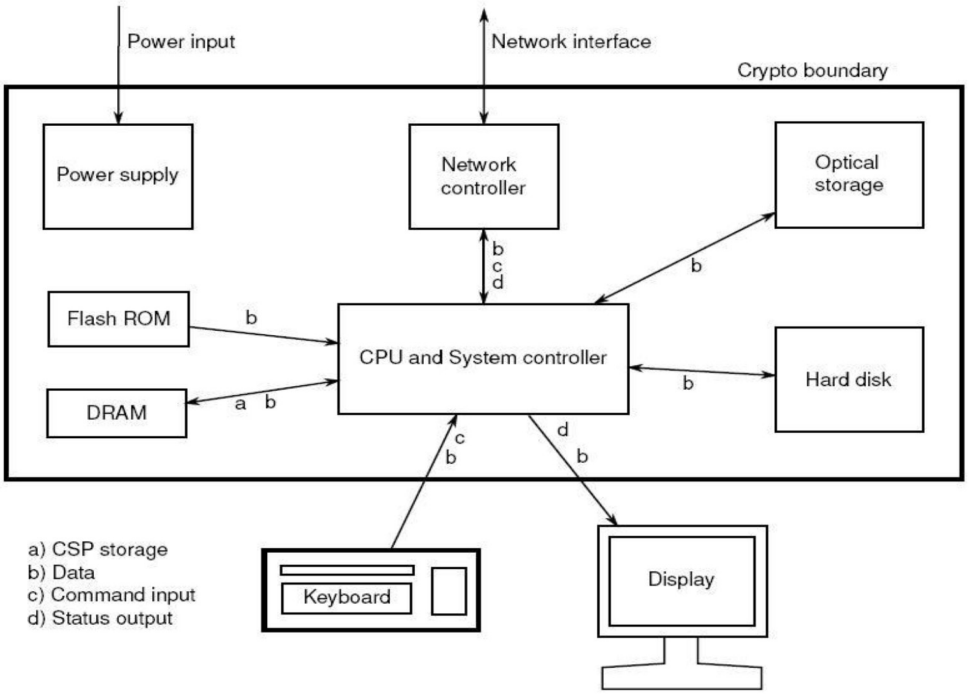

Figure 2: Hardware Block Diagram

2.2 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.

The module maintains separate contexts for each cryptographic operation. Therefore, critical security parameters (CSPs) used or stored in FIPS mode are not used in non-FIPS mode, and vice versa.

3 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 application program interface (API) through which applications request services. The following table summarizes the logical interfaces.

Logical Interface	Description
Data Input	API input parameters from kernel system calls, AF_ALG type socket.
Data Output	API output parameters from kernel system calls, AF_ALG type socket.
Control Input	API function calls, API input parameters from kernel system calls, AF_ALG type socket, kernel command line.
Status Output	API return values, AF_ALG type socket, kernel logs.

Table 4: Ports and Interfaces

4 Roles, Services and Authentication

4.1 Roles

The module meets all FIPS 140-2 level 1 requirements for Roles and Services, implementing both User and Crypto Officer role. The module does not allow concurrent operators.

- User role: performs all services, except module installation and configuration.
- Crypto Officer role: performs module installation and configuration.

The User and CO roles are implicitly assumed by the entity accessing the services implemented by the module. No authentication is required.

4.2 Services

The module provides services to the operators 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, it lists the associated cryptographic algorithm(s), the role to perform the service, the cryptographic keys or CSPs involved, and their access type(s). 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 encryption and decryption	AES	User	AES key	Read
	Three-key Triple-DES	User	Triple-DES key	Read
Random number generation	DRBG	User	Entropy input string, Internal state	Read, Update
Message digest	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	User	None	N/A
Message authentication code (MAC)	HMAC	User	HMAC key	Read
	CMAC with AES	User	AES key	Read
	CMAC with Triple-DES	User	Triple-DES key	Read
Encrypt-then-MAC (authenc) operation for IPsec	AES (CBC or CTR mode), HMAC	User	AES key, HMAC key	Read
	Triple-DES (CBC mode), HMAC		Triple-DES key, HMAC key	Read
Key encapsulation	RSA encrypt/decrypt primitives	User	RSA key pair	Read
Key wrapping	AES-CCM AES-GCM AES (ECB, CBC, CTR) and HMAC	User	AES key, HMAC key	Read
Shared Secret Computation	Diffie-Hellman EC Diffie-Hellman	User	Diffie-Hellman domain parameters	Read
			EC Diffie-Hellman key pair	Read
			Shared Secret	Read, Update

Service	Algorithm	Role	Keys/CSPs	Access
Zeroization	N/A	User	All CSPs	Zeroize
Self-tests	AES, Triple-DES, SHA-3, SHS, HMAC, DRBG, RSA signature verification	User	None	N/A
Other services				
Error detection code	crc32c[1], crct10dif[1]	User	None	N/A
Data compression	deflate[1], lz4[1], lz4hc[1], lzo[1], zlib[1], 842[1]	User	None	N/A
Memory copy operation	ecb(cipher_null)[1]	User	None	N/A
Show status	N/A	User	None	N/A
Module installation and configuration	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	Role	Keys	Access
Symmetric encryption and decryption	AES-XTS with 192-bit key size	User	AES key	Read
	Generic GCM encryption with external IV, RFC4106 GCM encryption with external IV	User	AES key	Read
Message digest	GHASH outside the GCM context	User	None	N/A
Message authentication code (MAC)	HMAC with less than 112-bit keys	User	HMAC key	Read
RSA signature generation	RSA sign primitive operation.	User	RSA key pair	Read
RSA signature verification	RSA verify primitive operation with keys smaller than 2048 bits.	User	RSA key pair	Read
RSA key encapsulation	RSA keys smaller than 2048 bits.	User	RSA key pair	Read
Key wrapping	AES-KW	User	AES key	Read
Shared Secret Computation	Diffie-Hellman with keys smaller than 2048 bits.	User	Diffie-Hellman domain parameters	Read
			Shared Secret	Read,

1 This algorithm does not provide any cryptographic attribute.

Service	Algorithm	Role	Keys	Access
				Update
EC key generation	EC key generation	User	EC key pair	Read Write

Table 6: Services in non-FIPS mode of operation

4.3 Operator Authentication

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

4.4 Algorithms

The module provides multiple implementations of algorithms. Different implementations can be invoked by using the unique algorithm driver names.

The module supports the use of generic C implementation of all algorithms; generic assembler for AES and Triple-DES algorithms; AES-NI for AES algorithm; CLMUL for GHASH algorithm used in GCM mode; AVX, AVX2 and SSSE3 for SHA algorithm; and multi-buffer implementation for the SHA-1, SHA-256 and SHA-512 algorithms.

Table 7 lists the approved algorithms, the CAVP certificates, and other associated information of the cryptographic implementations in FIPS mode.

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs.
AES	ECB	128, 192, 256	Data Encryption and Decryption	FIPS197, SP800-38A	A58, A66, A70 (generic C) A63, A68, A71 (constant time C for cipher) A57, A69, A72 (assembler for cipher) A55, A60, A61 (AESNI for cipher) A53, A54, A59 (AESNI for cipher, assembler for block chaining)
	CBC, CTR	128, 192, 256	Data Encryption and Decryption	FIPS197, SP800-38A	A66 (generic C) A68 (constant time C for cipher) A57 (assembler for cipher) A60 (AESNI for cipher) A53 (AESNI for cipher, assembler for block chaining)
	XTS	128, 256	Data Encryption and Decryption for Data Storage	SP800-38E	
	GCM with external IV	128, 192, 256	Data Decryption	SP800-38D	
	CMAC	128, 192, 256	MAC Generation and Verification	SP800-38B	A66 (generic C)

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs.
	CCM	128, 192, 256	Data Encryption and Decryption	SP800-38C	A68 (constant time C for cipher) A57 (assembler for cipher) A60 (AESNI for cipher)
	GMAC	128, 192, 256	Message authentication code	SP800-38D	
	RFC4106 GCM with internal IV	128, 192, 256	Data Encryption	SP800-38D RFC4106	A70 (generic C) A63 (constant time C for cipher) A69 (assembler for cipher) A55 (AESNI for cipher) A59 (AESNI for cipher, assembler for block chaining)
	RFC4106 GCM with external IV	128, 192, 256	Data Decryption	SP800-38D RFC4106	A58 (generic C) A71 (constant time C for cipher) A72 (assembler for cipher) A61 (AESNI for cipher) A54 (AESNI for cipher, assembler for block chaining)
DRBG	CTR_DRBG: AES-128, AES-192, AES-256 with derivation function	N/A	Deterministic Random Bit Generation	SP800-90A	A66 (generic C) A58 (constant time C for cipher) A57 (assembler for cipher) A60 (AESNI for cipher) A53 (AESNI for cipher, assembler for block chaining)
	Hash_DRBG: SHA-1, SHA-256, SHA-384, SHA-512 HMAC_DRBG: SHA-1, SHA-256, SHA-384, SHA-512	N/A	Deterministic Random Bit Generation	SP800-90A	A66 (generic C) A62 (use of AVX for SHA) A73 (use of AVX2 for SHA) A74 (use of SSSE3 for SHA) A58 (constant time C for cipher) A57 (assembler for

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs.
					cipher) A60 (AESNI for cipher) A53 (AESNI for cipher, assembler for block chaining)
HMAC	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message authentication code	FIPS198-1	A66 (generic C) A62 (use of AVX for SHA) A73 (use of AVX2 for SHA) A74 (use of SSSE3 for SHA)
	SHA3-224, SHA3-256, SHA3-384, SHA3-512	112 or greater	Message authentication code	FIPS198-1	A56
KAS ECCCDH component		P-256	Share secret computation	SP800-56A IG D.8	CVL A52
KAS FCC component	SHA-224, SHA-256	2048	Share secret computation	SP800-56A IG D.8	CVL A64
RSA	PKCS#1v1.5: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	2048, 3072, 4096	Digital Signature Verification	FIPS186-4	A66 (generic C) A62 (use of AVX for SHA) A73 (use of AVX2 for SHA) A74 (use of SSSE3 for SHA)
SHA-3	SHA3-224, SHA3-256, SHA3-384, SHA3-512	N/A	Message Digest	FIPS202	A56
SHS	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest	FIPS180-4	A66 (generic C) A62 (use of AVX) A73 (use of AVX2) A74 (use of SSSE3)
	SHA-1, SHA-256, SHA-512				A65 (use of multi-buffer)
Triple-DES (three-key)	ECB, CBC, CTR	192	Data Encryption and Decryption	SP800-67 SP800-38A	A66 (generic C) A57 (assembler for cipher) A67 (assembler for cipher and block

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs.
					chaining)
	CMAC	192	MAC Generation and Verification	SP800-67 SP800-38B	A66 (generic C) A57 (assembler for cipher)

Table 7: Approved Cryptographic Algorithms for Intel Xeon Processor

4.5 Non-Approved Algorithms

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

Algorithm	Use
NDRNG	The module obtains the entropy data from NDRNG to seed the DRBG
RSA encrypt/decrypt primitives with keys equal or larger than 2048 bits up to 15360 or more	Key Transport; allowed per [FIPS140-2_IG] D.8

Table 8: Non-Approved but Allowed Algorithms

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

Algorithm	Implementation name	Use
AES fixed-time implementation	"aes-fixed-time"	Data Encryption and Decryption
AES-XTS with 192-bit keys	"xts"	Data Encryption and Decryption
AES-KW	"kw(aes)"	Key Wrapping
Generic GCM encryption with external IV	“gcm(aes)” with external IV	Data Encryption
RFC4106 GCM encryption with external IV	“rfc4106(gcm(aes))” with external IV	Data Encryption
GHASH	"ghash"	Message Digest outside the GCM mode
HMAC with less than 112 keys	"hmac"	Message Authentication Code
EC Key Generation	“ecdh”	EC Key Generation
RSA sign primitive operation	"rsa"	Digital Signature Generation
RSA verify primitive operation with keys smaller than 2048 bits.	"rsa"	Digital Signature Verification
RSA encrypt/decrypt with keys smaller than 2048 bits	"rsa"	Key Encapsulation
Diffie-Hellman shared secret computation with keys other than 2048 bits	"dh"	Shared Secret Computation

Table 9: Non-Approved Cryptographic Algorithms

5 Physical Security

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

6 Operational Environment

This module operates in a modifiable operational environment per the FIPS 140-2 level 1 specifications.

6.1 Policy

The operating system shall be restricted to a single operator mode of operation (i.e., 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.

7 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	N/A	The keys are passed into the module via API input parameters in plaintext.	Zeroized when freeing the cipher handler.
Triple-DES keys			
HMAC keys			
Entropy input string	Obtained from NDRNG	N/A	Zeroized when freeing the cipher handler.
DRBG internal state: V value, C value, key (if applicable) and seed material	Derived from entropy input as defined in SP800-90A	N/A	Zeroized when freeing the cipher handler.
Diffie-Hellman domain parameters	N/A	The domain parameters are passed into the module via API input parameters.	Zeroized when freeing the cipher handler.
EC Diffie-Hellman key pair	N/A	The key is passed into the module via API input parameters in plaintext.	Zeroized when freeing the cipher handler.
Shared secret	Generated during the Diffie Hellman or EC Diffie Hellman shared secret computation.	N/A	Zeroized when freeing the cipher handler.
RSA Key Pair (for key encapsulation)	N/A	The key is passed into the module via API input parameters in plaintext.	Zeroized when freeing the cipher handler.
RSA Public Key (for signature verification)	N/A	The key is passed into the module via API input parameters in plaintext.	Zeroized when freeing the cipher handler.

Table 10: Life cycle of Keys or CSPs

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

7.1 Random Number Generation

The module employs a SP 800-90A DRBG as a random number generator for the creation of random numbers. In addition, the module provides a Random Number Generation service to applications.

The DRBG supports the Hash_DRBG, HMAC_DRBG and CTR_DRBG mechanisms. The DRBG is initialized during module initialization.

The module uses a Non-Deterministic Random Number Generator (NDRNG) as the entropy source. The NDRNG is based on the Linux RNG and the CPU time jitter RNG, both within the module's logical boundary.

For seeding and reseeding the DRBG, the module obtains an amount of random data from the NDRNG that is 1.5 times the security strength expected for the DRBG method (e.g. 384 bits

for the CTR_DRBG using AES-256). Therefore, the module ensures that the NDRNG always provides the required amount of entropy to meet the security strength of the DRBG methods during initialization (seed) and reseeding.

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

7.2 Key/CSP Generation

The module does not provide any dedicated key generation service. However, the Random Number Generation service can be called by the user to obtain random numbers that can be used as key material for symmetric algorithms (AES and Triple-DES) and HMAC.

7.3 Key Agreement / Key Transport / Key Derivation

The module provides shared secret computation for Diffie-Hellman and EC Diffie-Hellman key agreement schemes.

The module also provides the following key transport mechanisms:

- Key wrapping using AES-CCM or AES-GCM.
- Key wrapping using AES in CBC or CTR modes and HMAC.RSA key encapsulation using private key encryption and public key decryption primitives.

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 or AES-GCM provides between 128 and 256 bits of encryption strength.
- AES key wrapping using AES in CBC or CTR modes and HMAC provides between 128 and 256 bits of encryption strength.
- RSA key wrapping[2]: key establishment methodology provides provides between 112 and 256 bits of encryption strength.
- Diffie-Hellman: shared secret computation provides 112 bits of encryption strength.
- EC Diffie-Hellman: shared secret computation provides 128 bits of encryption strength.

7.4 Key/CSP Entry and Output

The module does not support manual key entry. It supports electronic entry of symmetric keys, HMAC keys and asymmetric keys via API input parameters in plaintext form.

7.5 Key/CSP Storage

The module does not perform persistent storage of keys. The keys and CSPs are stored as plaintext in the RAM. The only exceptions are the HMAC keys and the RSA public key used for the integrity tests, which are stored in the module and rely on the operating system for protection.

7.6 Key/CSP Zeroization

When a calling application calls the appropriate API function that operation overwrites the memory with zeros and deallocates the memory when the cipher handler is freed.

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

8 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).

9 Self Tests

The module performs both power-up self-tests at module initialization and conditional tests during operation to ensure that the module is not corrupted and that the cryptographic algorithms work as expected.

The services are only available when the power-up self-tests have succeeded. If the power-up self-tests pass, a success message is recorded in the dmesg and the module turns operational, being all crypto services available. If the power-up self-tests fail, the module outputs an error message and enters the error state.

On-demand self-tests can be invoked by rebooting the operating system.

Conditional tests are performed during the operation of the module. If a conditional test is successful, the module remains operational. If it fails, the module outputs an error message and enters the error state.

When the module is in the error state, data output is inhibited and no further operations are possible. The operating system must be rebooted.

Table 11 lists all the self-tests performed by the module. For algorithms that have more than one implementation in the module (per Table 7), the module performs self-tests independently for each of these implementations.

Self Test	Description
Power-up tests performed at power-up and on demand:	
Cryptographic Algorithm Known Answer Tests (KATs)	KATs for AES in ECB, CBC, CTR, GCM, CCM and XTS modes; encryption and decryption are performed separately. KATs for Triple-DES in ECB, CBC and CTR modes; encryption and decryption are performed separately. KATs for AES and Triple-DES CMAC generation. KATs for SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512. KATs for SHA3-224, SHA3-256, SHA3-384 and SHA3-512. KATs for HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384 and HMAC-SHA-512. KATs for HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384 and HMAC-SHA3-512. KATs for Hash_DRBG, HMAC_DRBG, and CTR_DRBG, with and without PR. KAT for RSA public key encryption and private key decryption with 2048 bit keys. KAT for RSA signature verification is covered by the integrity tests performed on kernel object files. KAT for Diffie-Hellman primitive "Z" computation with 2048-bit key. KAT for EC Diffie-Hellman primitive "Z" computation with P-256 curve.
Software Integrity Test	The module uses the HMAC-SHA-256 algorithm for the integrity test of the static kernel binary and the fipscheck application. The HMAC calculation is performed by the fipscheck application itself. The module uses RSA signature verification using SHA-256 with a 4096-bit key for the integrity test of each of the kernel object files loaded during boot-up time.
Conditional tests performed during operation:	
Continuous Random Number Generator Test (CRNGT)	The module performs conditional self-tests on the output of NDRNG to ensure that consecutive random numbers do not repeat. It also performs DRBG health tests as specified in section 11.3 of SP 800-90A.
On demand execution of self tests	
On Demand Testing	Invocation of the self tests on demand can be achieved by rebooting the operating system.

Table 11: Self-Tests

10 Guidance

10.1 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 Package	Contents
x86_64	kernel-default-4.12.14-150.47.x86_64.rpm	Static kernel binary, hmac file, and kernel object files.
	dracut-fips-044.2-18.35.1.x86_64.rpm	Used to configure the operational environment to support FIPS as stated in section 10.1.2.
	libkcapi-tools-0.13.0-1.114.x86_64.rpm	Integrity test utility and hmac file.

Table 12: RPM packages

Additional kernel components that register with the Kernel Crypto API must not be loaded as the kernel configuration is fixed in approved mode.

10.1.1 Module Installation

The Crypto Officer can install the RPM packages containing the module as listed in Table 12 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.

10.1.2 Operating Environment Configurations

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
```

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"
```

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.

10.2 User Guidance

10.2.1 Cipher References and Priority

The cryptographic module provides multiple implementations of different algorithms as shown in Section 4.4. For example, the module provides the following implementations of AES:

- AES implemented with C code when the aes-generic kernel component is loaded
- AES using AES-NI Intel instruction set when the aesni-intel kernel component is loaded
- AES implemented with generic assembler code when the aes-x86_64 kernel component is loaded

Note, if more than one of the above listed kernel components are loaded, the respective implementation can be requested by using the following cipher mechanism strings with the initialization calls (such as crypto_alloc_blkcipher):

- aes-generic kernel component: “aes-generic”
- aesni-intel kernel component: "__aes-aesni"
- aes-x86_64 kernel component: "aes-asm"

The AES cipher can also be loaded by simply using the string "aes" with the initialization call. In this case, the AES implementation whose kernel component is loaded with the highest priority is used. The following priority exists:

- aesni-intel
- aes-x86_64
- aes-generic

For example: If the kernel components aesni-intel and aes-asm are loaded and the caller uses the initialization call (such as crypto_alloc_blkcipher) with the cipher string of "aes", the aesni-intel implementation is used. On the other hand, if only the kernel components of aes-x86_64 and aes-generic are loaded, the cipher string of "aes" implies that the aes-x86_64 implementation is used.

The discussion about the naming and priorities of the AES implementation also applies when cipher strings are used that include the block chaining mode, such as "cbc(aes-asm)", "cbc(aes)", or "cbc(__aes-aesni)".

When using the module, the user shall utilize the Linux kernel crypto API provided memory allocation mechanisms. In addition, the user shall not use the function copy_to_user() on any portion of the data structures used to communicate with the Linux kernel crypto API.

10.2.2 AES XTS

As specified in SP800-38E, the AES algorithm in XTS mode is designed for the cryptographic protection of data on storage devices. Thus it can only be used for the disk encryption functionality offered by dm-crypt (i.e., the hard disk encryption scheme). For dm-crypt, the

length of a single data unit encrypted with AES XTS mode is at most 65536 bytes (64KB of data), which does not exceed 2^{20} AES blocks (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.

10.2.3 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.

Generation of the GCM IV is compliant with IG A.5, provision 1.b (“IPSec protocol IV generation”).

When a GCM IV is used for encryption, only the RFC4106 GCM with internal IV generation is in compliance with the IPSec specification and shall only be used for the IPSec protocol. This IV generation is compliant with RFC4106 and an IKEv2 protocol RFC7296 shall be used to establish the shared secret SKEYSEED from which the AES GCM encryption keys are derived.

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 IPSec protocol ensures that the nonce_explicit, or counter portion, of the IV will not exhaust all of its possible values.

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.

10.2.4 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.

10.3 Handling Self Test Errors

Self test failure within the kernel crypto API module will panic the kernel and the operating system will not load.

The module can return to operational state by rebooting the system. If the failure continues, you must re-install the software package and make sure to follow all instructions. If you downloaded the software please verify the package hash to confirm a proper download. Contact SUSE if these steps do not resolve the problem.

The kernel dumps self-test success and failure messages into the kernel message ring buffer. Post boot, the messages are moved to */var/log/messages*.

Use **dmesg** to read the contents of the kernel ring buffer. The format of the ringbuffer (**dmesg**) output is:

```
alg: self-tests for %s (%s) passed
```

Typical messages are similar to "alg: self-tests for xts(aes) (xts(aes-x86_64)) passed" for each algorithm/sub-algorithm type.

11 Mitigation of Other Attacks

No other attacks are mitigated.

Appendix A 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 B 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

PKCS#1 **Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1**
https://www.ietf.org/rfc/rfc3447.txt

RFC4106 **The Use of Galois/Counter Mode (GCM) in Ipsec Encapsulating Security Payload (ESP)**
https://tools.ietf.org/html/rfc4106

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**
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**
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
SP800-67	**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