SUSE Linux Enterprise Server 12 - Kernel Crypto API Cryptographic Module version 1.0

FIPS 140-2 Non-Proprietary Security Policy

Version 1.3

Last update: 2015-12-16

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 12 - Kernel Crypto API Cryptographic Module version 1.0. It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS 140-2 (Federal Information Processing Standards Publication 140-2) for a security level 1 module.

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

FIPS 140-2 details the requirements of the Governments of the U.S. and Canada for cryptographic modules, aimed at the objective of protecting sensitive but unclassified information.

For more information on the FIPS 140-2 standard and validation program please refer to the NIST website at http://csrc.nist.gov/.

Throughout the document “SUSE Linux Enterprise Server 12 - Kernel Crypto API Cryptographic Module”, “the kernel crypto API module”, or “the module” are used interchangeably to refer to the SUSE Linux Enterprise Server 12 - Kernel Crypto API Cryptographic Module.

1.2 Document Organization / Copyright

This non-proprietary security policy document may be reproduced and distributed only in its original entirety without any revision, ©2014 SUSE.

1.3 External Resources / References

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

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

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

2 Cryptographic Module Specification

2.1 Module Overview

The SUSE Linux Enterprise Server 12 - Kernel Crypto API Cryptographic Module is a software cryptographic module that provides general-purpose cryptographic services to the remainder of the Linux kernel. For FIPS 140-2 purposes, it is a software-only, multi-chip standalone cryptographic module validated at 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 shows the multi-chip standalone platforms on which the module has been tested:

Manufacture	Model	O/S & Ver.
HP	ProLiant DL320e Generation 8	SUSE Linux Enterprise Server 12 (with and without AES-NI)

Table 2: Tested Platforms

This cryptographic module consists of the following components:

- The Linux kernel with the version of the RPM file of 3.12.44-52.10
- The dracut-fips package with the version of the RPM file of 037-37.2. It provides the configuration of the FIPS mode.
- OpenSSL, a separately validated cryptographic module (FIPS 140-2 validation certificate #2435). It provides cryptographic algorithms and security functions for fipscheck application.
- The fipscheck package with the version of the RPM file of 1.2.0-9.3. It performs the integrity check of the Linux kernel binary. The fipscheck application uses the OpenSSL module in accordance with the security rules stated in the SUSE Linux Enterprise Server 12 - OpenSSL Module v2.0 Security Policy.

2.2 Modes of Operation

The module supports two modes of operation: FIPS approved and non-approved.

In FIPS approved mode, the kernel will support the approved cryptographic algorithms as shown in Table 3. Column four lists the CAVP validation numbers obtained from NIST for successful validation testing of the implementation of the cryptographic algorithms on the platform as shown in Table 2.

FIPS approved Algorithm	Standards	Usage / Description (refer to Table 6 for the available services in Approved mode)	Validation No.
Triple-DES	SP 800-67 SP 800-38A	- Encryption and decryption with all three independent keys in ECB, CBC, and CTR modes - CMAC with 3-key TDES	Cert. #1873
AES	FIPS 197 SP 800-38A SP 800-38B SP 800-38C SP 800-38D SP 800-38E	C implementation of AES: - Encryption and decryption with 128/192/256-bit key in ECB, CBC and CTR modes - CMAC and CCM with 128/192/256-bit key - XTS with 128/256-bit key (for storage application usage only)	Cert. #3288
		AES-NI assembler implementation: - Encryption and decryption with 128/192/256-bit key in ECB, CBC and CTR modes - CMAC and CCM with 128/192/256-bit key - XTS with 128/256-bit key (for storage application usage only)	Cert. #3286
		Generic assembler implementation of AES: - Encryption and decryption with 128/192/256-bit key in ECB, CBC and CTR modes - CMAC and CCM with 128/192/256-bit key - XTS with 128/256-bit key (for storage application usage only)	Cert. #3287
AES-GCM	SP 800-38D	C implementation of AES-GCM for IPSec as specified in RFC4106	Cert. #3297
		AES-NI implementation of AES-GCM for IPSec as specified in RFC4106	Cert. #3298
RSA	FIPS 186-4	PKCS#1 v1.5 signature verification with 2048/3072-bit modulus size using SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512	Cert. #1687
SHS	FIPS 180-4	C implementation (byte-only): SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512	Cert. #2727
		SSSE3 assembler implementation (byte-only): SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512	Cert. #2725
		AVX assembler implementation (byte-only): SHA-224, SHA-256, SHA-384, and SHA-512	Cert. #2724

FIPS approved Algorithm	Standards	Usage / Description (refer to Table 6 for the available services in Approved mode)	Validation No.
		AVX2 assembler implementation (byte-only): SHA-224, SHA-256, SHA-384, and SHA-512	Cert. #2726
Hash_DRBG	SP 800-90A	C implementation: SHA-1, SHA-256, SHA-384, and SHA-512	Cert. #747
		SSSE3 assembler implementation: SHA-1, SHA-256, SHA-384, and SHA-512	Cert. #745
		AVX assembler implementation: SHA-256, SHA-384, and SHA-512	Cert. #744
		AVX2 assembler implementation: SHA-256, SHA-384, and SHA-512	Cert. #746
HMAC_DRBG	SP 800-90A	C implementation: SHA-1, SHA-256, SHA-384, and SHA-512	Cert. #747
		SSSE3 assembler implementation: SHA-1, SHA-256, SHA-384, and SHA-512	Cert. #745
		AVX assembler implementation: SHA-256, SHA-384, and SHA-512	Cert. #744
		AVX2 assembler implementation: SHA-256, SHA-384, and SHA-512	Cert. #746
CTR_DRBG	SP 800-90A	C implementation: AES with 128/192/256-bit key	Cert. #747
		AES-NI assembler implementation: AES with 128/192/256-bit key	Cert. #744
		Generic assembler implementation: AES with 128/192/256-bit key	Cert. #745
HMAC	FIPS 198	C implementation: KS<BS, KS=BS, KS>BS SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512	Cert. #2086
		SSSE3 assembler implementation: KS<BS, KS=BS, KS>BS SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512	Cert. #2084
		AVX assembler implementation: KS<BS, KS=BS, KS>BS SHA-224, SHA-256, SHA-384, and SHA-512	Cert. #2083
		AVX2 assembler implementation: KS<BS, KS=BS, KS>BS SHA-224, SHA-256, SHA-384, and SHA-512	Cert. #2085

Table 3: Approved Algorithms

NOTE:

1. *The only AES-GCM implementations allowed in FIPS mode are the ones corresponding to ther RFC4106 implementations (CAVS certificates #3297 and #3298). The AES-GCM implementations corresponding to the CAVS certificate #3287 and #3288 are not approved in FIPS mode, although they have a CAVS certificate.*

The integrity check of the module is performed by the HMAC-SHA-256 algorithm, which is provided by a FIPS 140-2 validated SUSE Linux Enterprise Server 12 – OpenSSL Module v2.0 with certificate #2435.

Table 4 shows the non-approved algorithms that can only be used in non-approved mode:

Non-approved Algorithm	**Usage / Description (refer to Table 7 for the available services in non-Approved mode)**
Anubis	Encryption and decryption
ARC4	Encryption and decryption
Blowfish	Encryption and decryption
Camelia	Encryption and decryption
CAST5	Encryption and decryption
CAST6	Encryption and decryption
DES	Encryption and decryption
Fcrypt	Encryption and decryption
Khazad	Encryption and decryption
Salsa20	Encryption and decryption
SEED	Encryption and decryption
Serpent	Encryption and decryption
TEA	Encryption and decryption
XTEA	Encryption and decryption
XETA	Encryption and decryption
Twofish	Encryption and decryption
Two key Triple-DES	Encryption and decryption
CTS block chaining mode	Ciphertext stealing (CTS) with AES CBC mode.
GCM block chaining mode	GCM block chaining mode for AES (Certs. #3286, #3287 and #3288)
LRW block chaining mode	LRW block chaining mode with AES, Camelia, CAST6, Serpent, or Twofish
PCBC block chaining mode	PCBC block chaining mode with Fcrypt
XTS operation mode using AES-192 block cipher	Encryption and decryption Note: CAVS currently does not provide a means to test it.

Non-approved Algorithm	Usage / Description (refer to Table 7 for the available services in non-Approved mode)
MD4	Message digest with 128-bit digest size
MD5	Message digest with 128-bit digest size
Michael Mic keyed digest (IEEE 802.11i/TKIP)	Keyed digest
RIPEMD	Message digest with digest size 128, 160, 256, 320 bits
Tiger	Message digest with TGR128, TGR160, TGR192
Whirlpool	Message digest with WP256, WP384, WP512
AES XCBC keyed digest	Keyed digest
RSA signature verification	PKCS #1 v1.5 signature verification Modulus size: 1024-4096 bits in multiple of 32 bits not listed in Table 3.

Table 4: Non-Approved Algorithms

2.3 Cryptographic Boundary

2.3.1 Hardware Block Diagram

The physical boundary of the module is the surface of the case of the target platform. Figure 1 shows the hardware block diagram of the module including the processors, memory, internal power supply, power interface, data status and control paths, etc. The bold line surrounding the hardware components represents the module's physical cryptographic boundary. The hardware devices consist of standard integrated circuits and does not include any security-relevant, semi- or custom integrated circuits or other active electronic circuit elements.

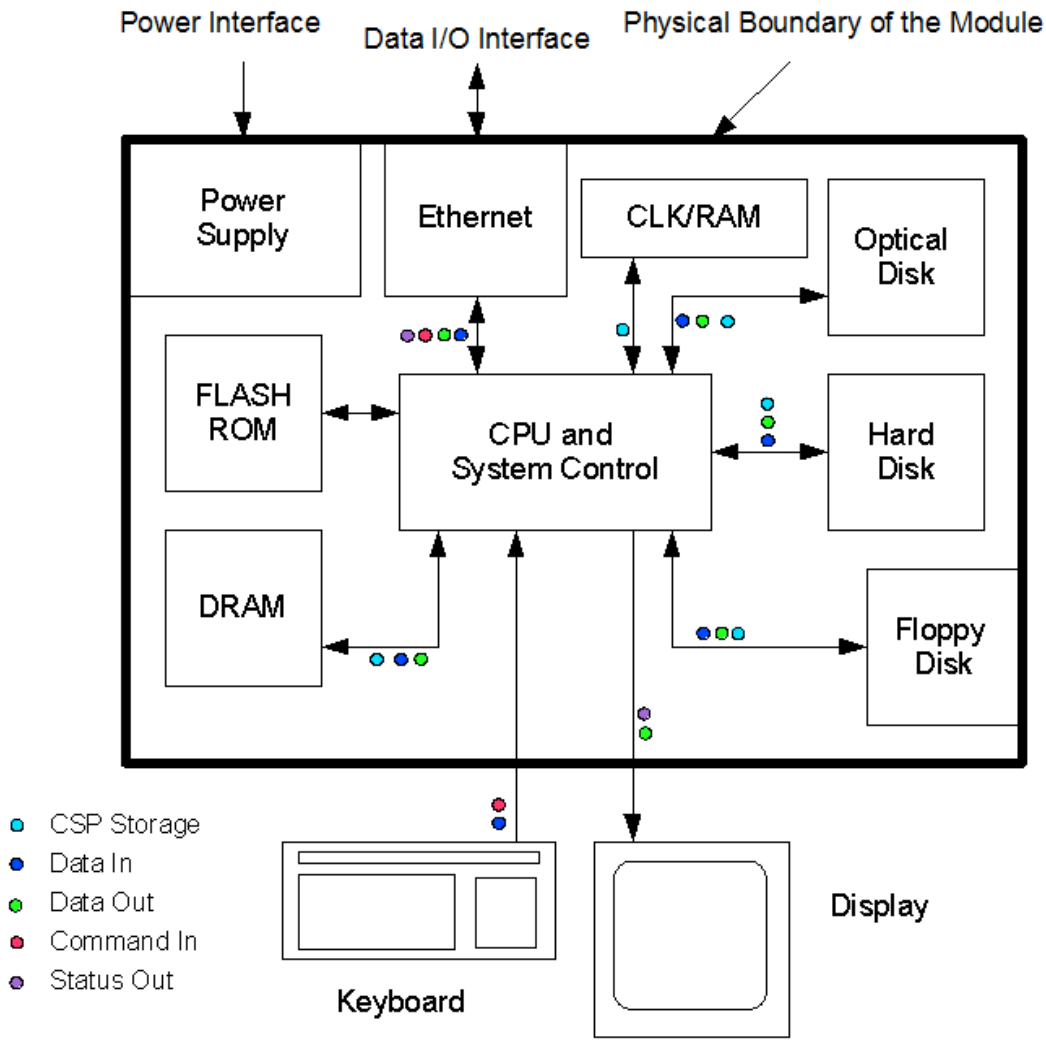

Figure 1: Hardware Block Diagram

2.3.2 Software Block Diagram

The logical boundary of the module is the binary code of the SUSE Linux Enterprise Server 12 - Kernel Crypto API Cryptographic Module itself, which is indicated by the “Logical Boundary of the Module” rectangle as illustrated in Figure 2. It consists of the Linux kernel cryptographic services and the fipscheck application. The kernel cryptographic services are implemented in the Linux kernel static binary /boot/vmlinuz-$(uname -r) file and in the directories of */lib/modules/$(uname -r)/kernel/crypto* and */lib/modules/$(uname -r)/kernel/arch/x86/crypto*.

The fipscheck application provides the integrity check mechanism during load time of the module and utilizes the HMAC-SHA-256 from the OpenSSL library from the SUSE Linux Enterprise Server 12 – OpenSSL Module v2.0 (Cert. #2435) which this module is bound to.

The IKEv2 protocol is implemented by the SUSE Linux Enterprise Server 12 - StrongSwan Cryptographic Module version 1.0 (Cert. #2484) which this module is bound to.

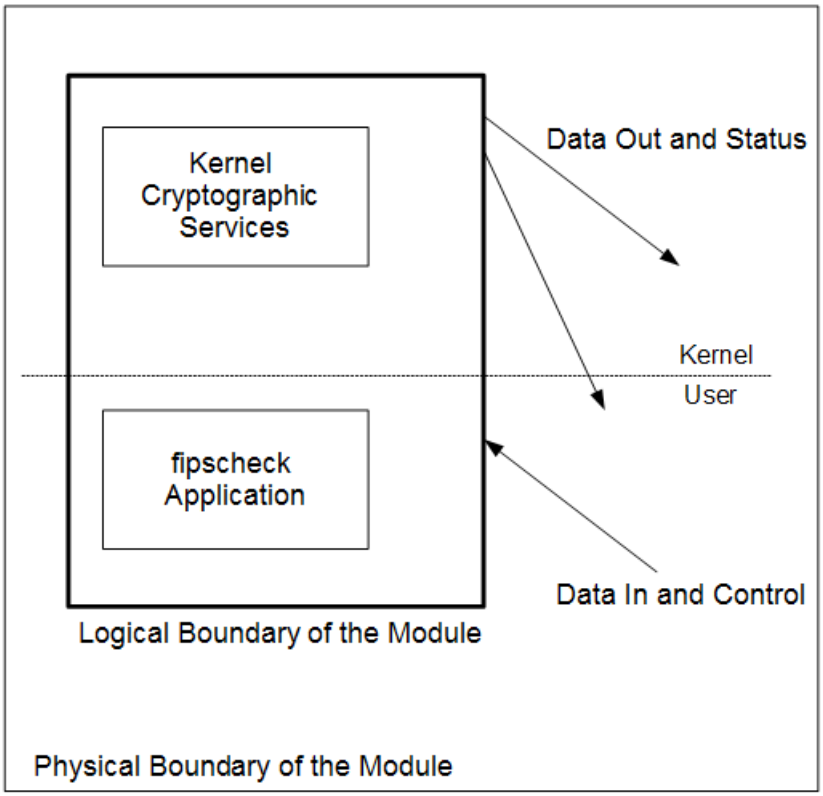

Figure 2: Software Block Diagram

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.

Module Logical Interface	Description
Data Input	API input parameters
Data Output	API output parameters
Control Input	API function calls
Status Output	API return values
Power Input	Physical power connector

Table 5: 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 (CO) role. The Module does not allow concurrent operators.

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

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

4.2 Services

The module provides services to authorized operators of either the User or CO role according to the applicable FIPS 140-2 security requirements.

Table 6 contains the services employed by the module in the approved mode. For each available service, it lists the associated role(s), the critical security parameters (CSPs) and cryptographic keys involved, and the type(s) of access to the CSPs and cryptographic keys.

CSPs contain security-related information whose disclosure or modification can compromise the main security objective of the module, namely the protection of sensitive information.

The access types are denoted as follows:

- 'R': The item is read or referenced by the service
- 'W': The item is written or updated by the service
- 'Z': The persistent item is zeroized by the service

Services	Roles		CSPs & crypto keys	Access Type
	User	CO		
Triple-DES encryption and decryption (- ECB, CBC, CTR and CMAC modes with 3 independent keys) Encryption: *Input:* plaintext, IV, key *Output:* ciphertext Decryption: *Input:* ciphertext, IV, key *Output:* plaintext	✓		Triple-DES key	R
AES encryption and decryption (- ECB, CBC, CTR, CMAC, CCM with 128, 192 or 156 bit keys - XTS with 128 or 256 bit keys - GCM mode implementation according to RFC4106 with 128, 192 or 256 bit keys) Encryption: *Input:* plaintext, IV, key *Output:* ciphertext Decryption: *Input:* ciphertext, IV, key	✓		AES key	R

Services	Roles		CSPs & crypto keys	Access Type
	User	CO		
Output: plaintext				
RSA signature verification *Input:* the modulus n, the public key e, the SHA algorithm, a message m, and a signature for the message *Output:* pass if the signature is valid, fail if the signature is invalid	✓		RSA public key	R
Secure Hash Generation *Input:* message *Output:* message digest	✓		none	N/A
DRBG random number generation *Input:* entropy, nonce, personalization string, and additional input *Output:* pseudorandom bits	✓		Seed, entropy input string, nonce, and intermediate values	R W Z
HMAC generation *Input:* HMAC key, message *Output:* HMAC value of message	✓		HMAC key	R W Z
HMAC-SHA-256 from OpenSSL for integrity check	✓		HMAC key	R W Z
Release all resources of symmetric cryptographic function context *Input:* context *Output:* N/A	✓		AES / Triple-DES key	Z
Release all resources of hash context *Input:* context *Output:* N/A	✓		HMAC key	Z
Self-tests	✓		HMAC key	R
Show status	✓		none	N/A
Install and initiate module		✓	none	N/A

Table 6: Roles and Approved Services Details

Services	Roles		Access Type
	User	CO	
Symmetric encryption and decryption (- non-Approved algorithms and non-Approved block chaining modes listed in Table 4: Anubis, ARC4, Blowfish, Camelia, CAST5, CAST6, DES, Fcrypt, Khazad, Salsa20, SEED, Serpent, TEA, XTEA, XETA, Twofish, Two key Triple-DES, CTS block chaining mode, GCM block chaining mode, LRW block chaining mode, PCBC block chaining mode, XTS operation mode using AES-192 block cipher)	✓		R
RSA signature verification (- modulus other than 2048 and 3072 bits)	✓		R
Secure Hash Generation (- non-Approved hash algorithms listed in Table 4: MD4, MD5, RIPEMD, Tiger, Whirlpool)	✓		N/A
Keyed Digest Generation (- non-Approved keyed digest listed in Table 4: AES XCBC keyed digest, Michael Mic)	✓		R

Table 7: Roles and non-Approved Services Details

The module provides a number of non-security relevant services to callers within kernel space as well as user space. The following documents provide a description of these services:

- Linux system call API man pages provided in chapter 2 of the Linux man pages obtainable from https://github.com/mkerrisk/man-pages
- Linux kernel internals including interfaces between kernel components documented in the book ISBN-13: 978-0470343432
- Linux kernel driver development documentation covering the kernel interfaces available for device drivers: ISBN-13: 978-0596005900

The non-security relevant services actually available will be based on the installed configuration.

4.3 Authentication

The module does not implement authentication. The role is implicitly assumed on entry.

4.4 Mechanism and Strength of Authentication

The module does not implement authentication.

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 definition.

6.1 Policy

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

The kernel component that makes calls to the cryptographic module is the single user of the cryptographic module, even when the application is serving multiple clients.

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

7.1 Random Number Generation

The module has a random number generator (RNG) that returns keying material for use in constructing cryptographic keys. It employs deterministic random bit generator (DRBG) mechanisms based on hash functions (Hash_DRBG and HMAC_DRBG), and on block cipher (CTR_DRBG) as specified in SP 800-90A.

The entropy and nonce is automatically obtained from the kernel-internal non-deterministic random number generator /dev/urandom, without user intervention. The DRBG automatically reseeds upon reaching the reseed threshold defined in SP 800-90A by obtaining data from the kernel-internal non-deterministic random number generator /dev/urandom .

7.2 Key/CSP Generation

The approved RNG specified in section 7.1 is used to generate cryptographic secret keys for symmetric crypto algorithms (AES and Triple-DES) and message authentication (HMAC).

The module does not output any information or intermediate results during the key generation process. The RNG itself is single-threaded.

7.3 Key/CSP Entry and Output

All keys are imported from, or output to, the invoking application running on the same device. The cryptographic module supports electronic entry of symmetric keys, RSA public key and HMAC keys via API parameters. The application using the module can pass secret key to the module in plain text form within the physical boundary. Keys are output from the module in plain text form if required by the calling application. The same holds for the CSPs.

7.4 Key/CSP Storage

With respect to the module, all keys are considered ephemeral. Any storage of keys is the responsibility of the calling program and is outside the scope of the module. All the keys and CSPs are stored within the kernel memory.

7.5 Key/CSP Zeroization

When a calling kernel components calls the appropriate API function that operation overwrites the memory with 0s and deallocates the memory (please see the API document for full details).

8 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The test platform as shown in Table 2 is compliant to 47 CFR FCC Part 15, Subpart B, Class A (Business use).

9 Self Tests

9.1 Power-Up Tests

The module performs both power-up self-tests at module initialization and continuous condition 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. No specific message is prompted when a module is successfully inserted in the kernel. If the power-up self-tests fail, the module will output an error message and enters to an error state. No further operations are possible when the module is in an error state.

The CO with physical or logical access to the module can run the power-up self-tests on demand by rebooting the operating system.

Self Test	Description
Mandatory power-up tests performed at power-up and on demand:	
Cryptographic Algorithm Known Answer Tests (KATs)	KATs for AES encryption and decryption are performed separately. KATs for Triple-DES encryption and decryption are performed separately. KATs for SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 are performed. KATs for HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512 are performed. KATs for Hash_DRBG, HMAC_DRBG, and CTR_DRBG are performed. KAT for RSA signature verification is performed by loading kernel modules and verifying their signature
Software Integrity Test	The module uses the HMAC-SHA-256 message authentication code (provided by the OpenSSL library) for the integrity test of the kernel static binary file */boot/vmlinuz-$(uname -r)*. The HMAC calculation is performed by the fipscheck application, and the value is stored in */boot/.vmlinuz-$(uname -r).hmac*. For all the other binary files in the directories */lib/modules/$(uname -r)/kernel/crypto* and */lib/modules/$(uname -r)/kernel/arch/x86/crypto*, an RSA signature verification of the each file is performed.
Conditional tests performed, as needed, during operation:	
DRBG	The module performs a continuous RNG test, whenever Hash_DRBG, HMAC_DRBG or CTR_DRBG is invoked. In addition, the seed source implemented in the operating system kernel also performs a continuous self test.
	The module performs a subset of the assurance tests as specified in section 11 of SP 800-90A, in particular it complies with the mandatory documentation requirements and performs KATs and prediction resistance.
On demand execution of self tests	
On Demand Testing	Invocation of the self tests on demand can be achieved by restarting the O/S.

Table 8: Self-Tests

10 Guidance

Password-based encryption and password-based key generation do not provide sufficient strength to satisfy FIPS-2 requirements. As a result, data processed with password-based encryption methods are considered to be unprotected.

10.1 Crypto Officer Guidance

Additional kernel modules that register with the kernel crypto API must not be loaded as the kernel configuration is fixed in approved mode.

To operate the module, the operating system must be restricted to a single operator mode of operation.

Secure installation and Startup:

Crypto officers use the Installation instructions to install the module in their environment.

The version of the RPM containing the validated module is stated in section 2 above. The integrity of the RPM is automatically verified during the installation and the crypto officer shall not install the RPM file if the RPM tool indicates an integrity error.

To bring the module into FIPS approved mode, perform the following:

1. Install the dracut-fips package:

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

2. Install the patterns-sles-fips package

```
# zypper in -t pattern fips
```

3. Recreate the INITRAMFS image:

```
# dracut -f
```

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 to the kernel command line:

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

10.2 User Guidance

The cryptographic module provides multiple implementations of AES given in the following bullet

- AES implemented with C code when the aes-generic kernel module is loaded
- AES using AES-NI Intel instruction set when the aesni-intel kernel module is loaded (the kernel module can only be used if the underlying processor supports the AES-NI instruction set)

- AES implemented with g assembler code when the aes-x86_64 kernel module is loaded

Note, if more than one of the above listed kernel modules 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 module: “aes-generic”
- aesni-intel kernel module: "__aes-aesni"
- aes-x86_64 kernel module: "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 module is loaded with the highest priority is used. The following priority exists:

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

For example: If the kernel modules 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 modules 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)".

The module maintains a process flag to indicate that the module is in FIPS 140-2 approved mode. The flag is provided in the file */proc/sys/crypto/fips_enabled*. If this file contains a value of 1, the module is operational in FIPS 140-2 approved mode

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.

Only the cryptographic mechanisms provided with the Linux kernel crypto API are considered to be used. The OpenSSL library, although used, is only considered to support the integrity check and is not intended for general-purpose use with respect to this cryptographic module.

The AES-XTS mode shall only be used for storage applications only and shall not be used for the encryption of data in transit.

To use AES GCM according to RFC4106 in FIPS mode, the user shall use the specific cipher mechanism string “rfc4106(gcm(aes))”, as the other implementations of AES GCM block chaining mode are not allowed in FIPS mode.

The IV generation method used in AES GCM implementation with Certs. #3297 and #3298 is in compliance with the IPSec-v3 protocol as described in RFC 6071 and the AES GCM encryption keys are derived from the shared secret SKEYSEED established by an IKEv2 protocol as described in RFC 7296. The IKEv2 is implemented in SUSE Linux Enterprise Server 12 – StrongSwan Cryptographic Module version 1.0 which is a FIPS validated module (Cert. #2484).

10.3 Handling Self Test Errors

Self test or continuous 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 crypto API module performs power-up self-tests that includes an integrity check and known-answer tests for the available cryptographic algorithms. It also runs conditional tests on the DRBG whenever it is used.

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.Mitigation of Other Attacks

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
O/S	Operating System
PKCS	Public Key Cryptography Standards
RNG	Random Number Generator
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 PUB **Security Requirements for Cryptographic Modules**
January 2011
http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

FIPS180-4 **Secure Hash Standard (SHS)**
March 2012
http://csrc.nist.gov/publications/fips/fips180-4/fips 180-4.pdf

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

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

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

PKCS#1 **Public Key Cryptography Standards (PKCS) #1: RSA Cryptography**
Specifications Version 2.1
February 2003
http://www.ietf.org/rfc/rfc3447.txt

SP800-38A **NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques**
December 2001
http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

SP800-38B **NIST Special Publication 800-38B - Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication**
May 2005
http://csrc.nist.gov/publications/nistpubs/800-38B/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
http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated July20_2007.pdf

SP800-38D **NIST Special Publication 800-38D - Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC**
November 2007
http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.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
http://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf

SP800-67 **NIST Special Publication 800-67 Revision 1 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher**
January 2012
http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf

SP800-90A **NIST Special Publication 800-90A - Recommendation for Random Number Generation Using Deterministic Random Bit Generators**
January 2012
http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf