SUSE Linux Enterprise Server - Kernel Crypto API Cryptographic Module version 2.0

FIPS 140-2 Non-Proprietary Security Policy

Doc version 2.3

Last update: 2017-12-23

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 2.0. It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS 140-2 (Federal Information Processing Standards Publication 140-2) for a security level 1 module.

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

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

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

Throughout the document, “the Kernel Crypto API module” and “the module” are also used to refer to the SUSE Linux Enterprise Server - Kernel Crypto API Cryptographic Module. The current version of the module is 2.0. An earlier version of this module has gone through FIPS 140-2 validation under certificate #2549.

This module is bound to the SUSE Linux Enterprise Server OpenSSL Module, validated under FIPS 140-2 Cert. #3038.

1.2 Document Organization / Copyright

This non-proprietary security policy document may be reproduced and distributed only in its original entirety without any revision, ©2017 SUSE, LLC / atsec information security.

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 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-$(uname -r)
Integrity check HMAC file for Linux kernel static binary	/boot/.vmlinuz-$(uname -r).hmac
Cryptographic kernel object files	On x86_64 system: /lib/modules/$(uname -r)/kernel/crypto/*.ko /lib/modules/$(uname -r)/kernel/arch/x86/crypto/*.ko On z system: /lib/modules/$(uname -r)/kernel/crypto/*.ko /lib/modules/$(uname -r)/kernel/arch/s390/crypto/*.ko
Integrity test utility	/usr/lib64/libkcapi/fipscheck
Integrity check HMAC file for	/usr/lib64/libkcapi/.fipscheck.hmac

Description	Component
integrity test utility	

Table 2: Cryptographic Module Components

The module uses the SUSE Linux Enterprise Server OpenSSL Module as a bound module (also referred to as “the bound OpenSSL module”), which provides the underlying cryptographic algorithms necessary for performing the integrity test of the Linux kernel static binary (the integrity test of the kernel objects is provided by the module itself). The SUSE Linux Enterprise Server OpenSSL Module is a FIPS-validated module with certificate #3038.

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

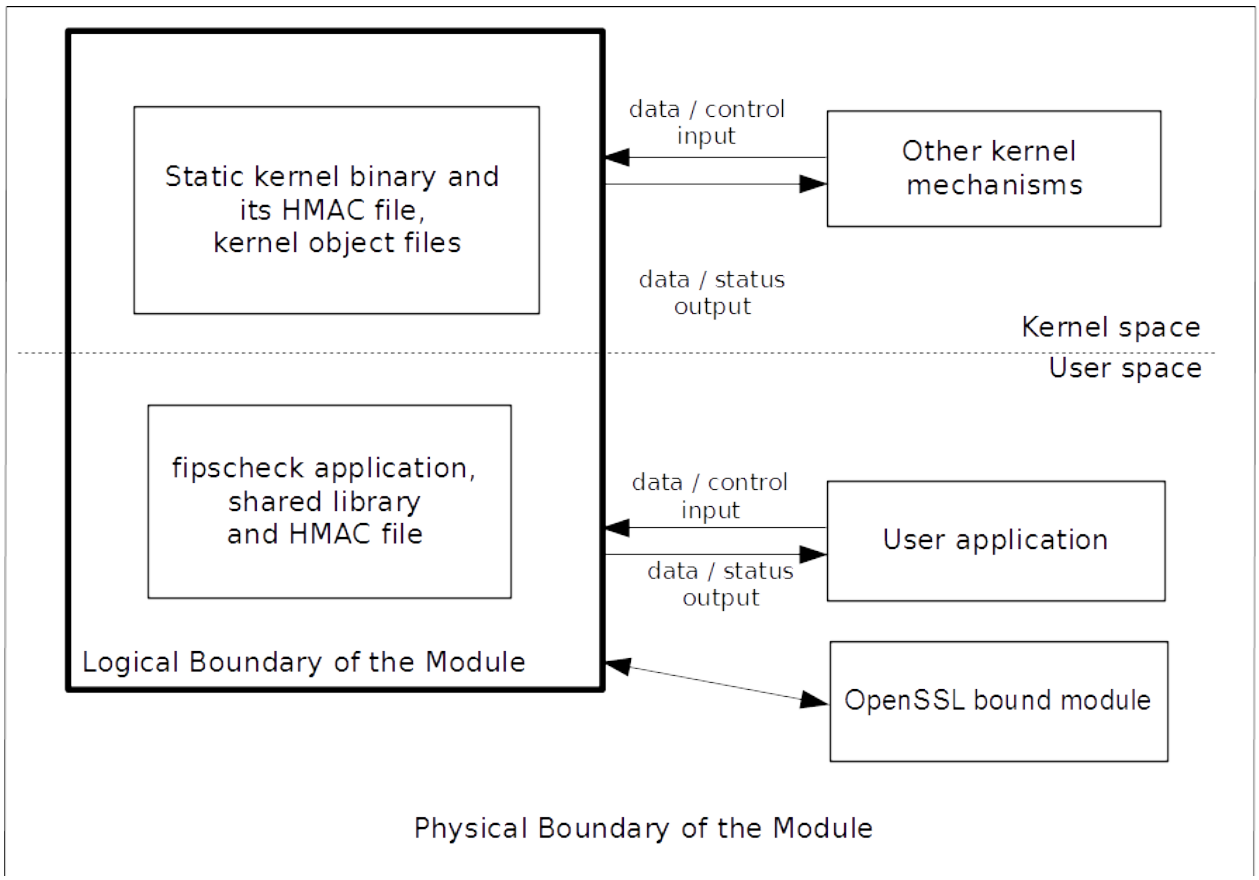

Figure 1: Software Block Diagram

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
FUJITSU Server PRIMERGY CX2570 M2 inside a CX400 M1 enclosure	Intel Xeon E5 family	SUSE Linux Enterprise Server 12 SP2 with and without AES-NI (PAA)
IBM z13	z13	SUSE Linux Enterprise Server 12 SP2 with and without CPACF (PAI)

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.

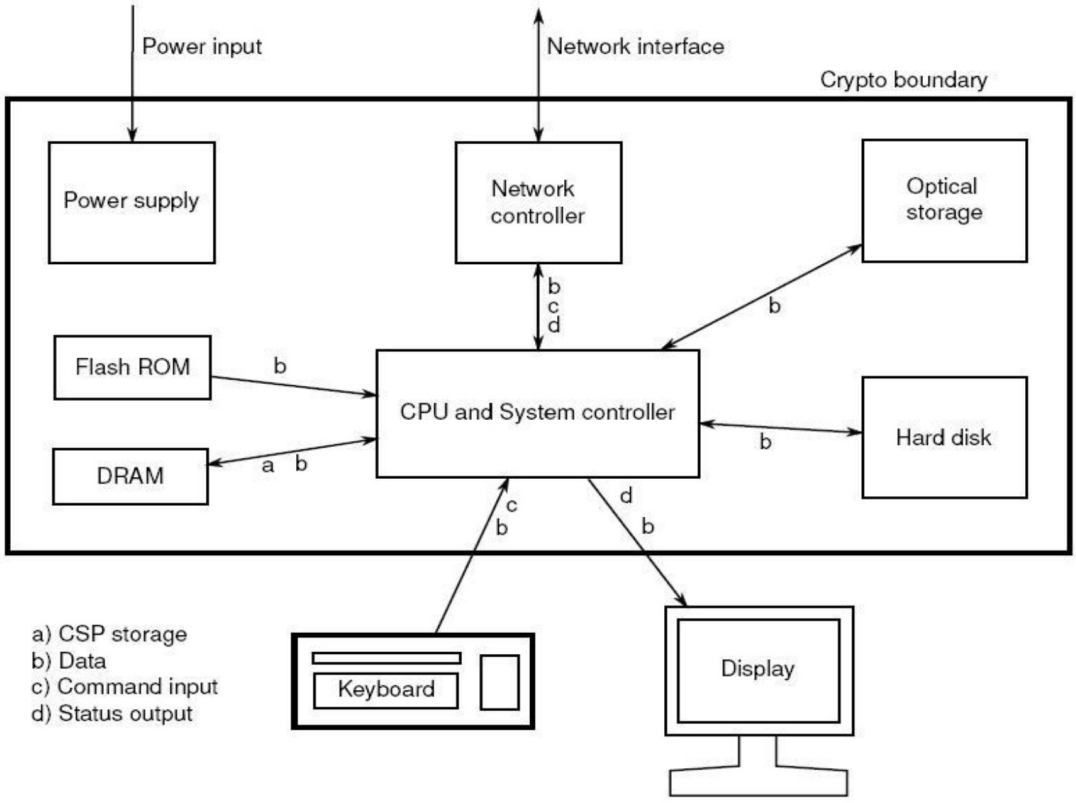

Figure 2: Hardware Block Diagram

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.

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.

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

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 further 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 CAVS certificate numbers can be found in Table 7 and Table 8.

Service	Algorithm	Role	Key/CSP	Access
Cryptographic services				
Symmetric encryption and decryption	AES	User	AES key	Read
	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	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) and HMAC, Triple-DES (CBC mode) and HMAC	User	AES/Triple-DES key, HMAC key	Read
AES key wrapping	AES	User	AES key	Read
Zeroization	N/A	User	All CSPs	Zeroize
Self-tests	AES, Triple-DES, SHS, HMAC, DRBG, RSA signature verification	User	None	Read
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

1 This algorithm does not provide any cryptographic attribute.

Service	Algorithm	Role	Key/CSP	Access
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 10.

Service	Algorithm	Role	Key/CSP	Access
Symmetric encryption and decryption	Anubis, ARC4, Blowfish, Camelia, CAST5, CAST6, ChaCha20, DES, Fcrypt, Khazad, Salsa20, SEED, Serpent, TEA, XETA, XTEA, Twofish, Two-key Triple-DES, CTS mode, generic GCM encryption with external IV, RFC4106 GCM encryption with external IV, LRW mode, PCBC mode	User	Symmetric key	Read
Message digest	MD4, MD5, Poly1305, RIPEMD, Tiger, Whirlpool	User	None	N/A
Message authentication code (MAC)	Michael Mic, XCBC	User	MAC key	Read
RSA primitive operations	RSA primitive operations for encryption, decryption, sign and verify	User	RSA key pair	Read

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.

4.4.1 Running on Intel Xeon Processor

On the platform that runs the Intel Xeon processor, the module supports the use of generic C implementation of all the 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 SHA-1 algorithm.

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

Algorithm	CAVS Cert.	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
AES	#4608 (C implementation of AES)	FIPS197, SP800-38A	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption
		SP800-38B	CMAC	128, 192, 256	MAC Generation and Verification

Algorithm	CAVS Cert.	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
	#4611 (generic assembler for AES core) #4614 (AES_NI for AES core and CLMUL for GHASH)	SP800-38C	CCM	128, 192, 256	Data Encryption and Decryption
		SP800-38D	Generic GCM with external IV	128, 192, 256	Data Decryption
		SP800-38E	XTS	128, 256	Data Encryption and Decryption for Data Storage
		SP800-38F	KW	128, 192, 256	Key Wrapping and Unwrapping
	#4609 (RFC4106 GCM with internal IV using C implementation of AES) #4612 (RFC4106 GCM with internal IV using generic assembler for AES core) #4615 (RFC4106 GCM with internal IV using AES_NI for AES core and CLMUL for GHASH)	FIPS197 SP800-38A	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption
		SP800-38D RFC4106	RFC4106 GCM with internal IV	128, 192, 256	Data Encryption
	#4610 (RFC4106 GCM with external IV using C implementation of AES) #4613 (RFC4106 GCM with external IV using generic assembler for AES core) #4616 (RFC4106 GCM with external IV using AES_NI for AES core and CLMUL for GHASH)	FIPS197 SP800-38A	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption
		SP800-38D RFC4106	RFC4106 GCM with external IV	128, 192, 256	Data Decryption
	#4617 (AES_NI for AES and RFC4106 GCM with external IV)	FIPS197 SP800-38A	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption
		SP800-38D RFC4106	RFC4106 GCM with external IV	128, 192, 256	Data Decryption
		SP800-38E	XTS	128, 256	Data Encryption

Algorithm	CAVS Cert.	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
					and Decryption for Data Storage
	#4618 (AES_NI for AES and RFC4106 GCM with internal IV)	FIPS197 SP800-38A	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption
		SP800-38D RFC4106	RFC4106 GCM with internal IV	128, 192, 256	Data Encryption
DRBG	#1544 (using C implementation of AES) #1545 (using generic assembler for AES core) #1546 (using AES_NI for AES core)	SP800-90A	CTR_DRBG: AES-128, AES-192, AES-256 with derivation function	N/A	Deterministic Random Bit Generation
	#1547 (using AVX for SHA) #1549 (using AVX2 for SHA) #1550 (using C implementation of SHA) #1551 (using SSSE3 for SHA)	SP800-90A	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
HMAC	#3054 (using AVX for SHA) #3056 (using AVX2 for SHA) #3057 (using C implementation of SHA) #3058 (using SSSE3 for SHA)	FIPS198-1	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	112 or greater	Message authentication code
RSA	#2513 (using AVX for SHA) #2515 (using AVX2 for SHA) #2516 (using C implementation of SHA) #2517 (using	FIPS186-4	PKCS#1v1.5: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024 or greater	Digital Signature Verification for integrity tests of kernel object files.

Algorithm	CAVS Cert.	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
	SSSE3 for SHA)				
SHS	#3782 (using AVX for SHA) #3784 (using AVX2 for SHA) #3785 (using C implementation of SHA) #3786 (using SSSE3 for SHA) #3787 (using multi-buffer for SHA-1[2])	FIPS180-4	SHA-1[2], SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest
Triple-DES (three-key)	#2452 (C implementation of TDES)	SP800-67 SP800-38A	ECB, CBC, CTR	192	Data Encryption and Decryption
		SP800-67 SP800-38B	CMAC with three-key Triple-DES	192	MAC Generation and Verification
	#2453 (generic assembler for TDES core) #2454 (generic assembler for TDES)	SP800-67 SP800-38A	ECB, CBC, CTR	192	Data Encryption and Decryption
OpenSSL (Cert. #3038) bound module algorithms					
HMAC-SHA-256	#3042 (AVX2 implementation) #3043 (AVX implementation) #3044 (SSSE3 implementation) #3045 (assembler implementation)	FIPS198-1		256	Integrity test of the kernel static binary
SHA-256	#3768 (AVX2 implementation) #3769 (AVX implementation) #3770 (SSSE3 implementation) #3771 (SSSE3	FIPS 180-4		N/A	Underlying SHA of the HMAC used for the Integrity test of the kernel static binary

2 Only SHA-1 is supported and tested using multi-buffer for SHA-1 implementation.

Algorithm	CAVS Cert.	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
	implementation) (assembler implementation)				

Table 7: Cryptographic Algorithms for Intel Xeon Processor

4.4.2 Running on z13 Processor

On the platform that runs the z system, the module supports the use of generic C implementation for all the algorithms, and the use of CPACF for AES, Triple-DES, GHASH and SHA algorithms.

Table 8 lists the approved algorithms, the CAVS certificates, and other associated information of the cryptographic implementations in FIPS mode.

Algorithm	CAVS Cert.	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
AES	#4681 (C implementation of AES) #4682 (CPACF for AES core)	FIPS197, SP800-38A	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption
		SP800-38B	CMAC	128, 192, 256	MAC Generation and Verification
		SP800-38C	CCM	128, 192, 256	Data Encryption and Decryption
		SP800-38D	Generic GCM with external IV	128, 192, 256	Data Decryption
		SP800-38E	XTS	128, 256	Data Encryption and Decryption for Data Storage
		SP800-38F	KW	128, 192, 256	Key Wrapping and Unwrapping
	#4677 (CPACF for AES and GHASH)	FIPS197 SP800-38A	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption
		SP800-38C	CCM	128, 192, 256	Data Encryption and Decryption
		SP800-38D RFC4106	Generic GCM with external IV	128, 192, 256	Data Decryption
		SP800-38E	XTS	128, 256	Data Encryption and Decryption for Data Storage
	#4672 (RFC4106 GCM with internal IV using C implementation of AES) #4674 (RFC4106 GCM with internal IV using CPACF for AES core)	FIPS197 SP800-38A	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption
		SP800-38D RFC4106	RFC4106 GCM with internal IV	128, 192, 256	Data Encryption

Algorithm	CAVS Cert.	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
	#4678 (RFC4106 GCM with internal IV using CPACF for AES)				
	#4673 (RFC4106 GCM with external IV using C implementation of AES) #4675 (RFC4106 GCM with external IV using CPACF for AES core) #4679 (RFC4106 GCM with external IV using CPACF for AES)	FIPS197 SP800-38A	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption
		SP800-38D RFC4106	RFC4106 GCM with external IV	128, 192, 256	Data Decryption
DRBG	#1582 (using CPACF for AES) #1584 (using C implementation of AES) #1585 (using CPACF for AES core)	SP800-90A	CTR_DRBG: AES-128, AES-192, AES-256 with derivation function	N/A	Deterministic Random Bit Generation
	#1585 (using CPACF for SHA) #1586 (using C implementation of SHA)	SP800-90A	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
HMAC	#3097 (using CPACF for SHA) #3098 (using C implementation of SHA)	FIPS198-1	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	112 or greater	Message authentication code
RSA	#2554 (using CPACF for SHA) #2555 (using C implementation of SHA)	FIPS186-4	PKCS#1v1.5: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024 or greater	Digital Signature Verification for integrity tests of kernel object files.
SHS	#3832 (CPACF for SHA)	FIPS180-4	SHA-1, SHA-224, SHA-256,	N/A	Message Digest

Algorithm	CAVS Cert.	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
	#3833 (C implementation of SHA)		SHA-384, SHA-512		
Triple-DES (three-key)	#2486 (CPACF for TDES)	SP800-67 SP800-38A	ECB, CBC, CTR	192	Data Encryption and Decryption
	#2487 (C implementation of TDES) #2489 (CPACF for TDES core)	SP800-67 SP800-38A	ECB, CBC, CTR	192	Data Encryption and Decryption
		SP800-67 SP800-38B	CMAC with three-key Triple-DES	192	MAC Generation and Verification
OpenSSL (Cert. #3038) bound module algorithms					
HMAC-SHA-256	#3059 (assembler implementation) #3060 (CPACF implementation)	FIPS198-1		256	Integrity test of the kernel static binary
SHA-256	#3788 (assembler implementation) #3789 (CPACF implementation)	FIPS 180-4		N/A	Underlying SHA of the HMAC used for the Integrity test of the kernel static binary

Table 8: Cryptographic Algorithms for z System z13 Processor

4.4.3 Non-Approved Algorithms

Table 9 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

Table 9: Non-Approved but Allowed Algorithms

Table 10 shows the non-Approved cryptographic algorithms implemented in the module that are only available in non-FIPS mode.

Algorithm	Use
Anubis	Data Encryption and Decryption
ARC4	Data Encryption and Decryption
Blowfish	Data Encryption and Decryption
Camelia	Data Encryption and Decryption
CAST5	Data Encryption and Decryption
CAST6	Data Encryption and Decryption

Algorithm	Use
ChaCha20	Data Encryption and Decryption
DES	Data Encryption and Decryption
Fcrypt	Data Encryption and Decryption
Khazad	Data Encryption and Decryption
Salsa20	Data Encryption and Decryption
SEED	Data Encryption and Decryption
Serpent	Data Encryption and Decryption
TEA	Data Encryption and Decryption
XETA	Data Encryption and Decryption
XTEA	Data Encryption and Decryption
Twofish	Data Encryption and Decryption
Two-key Triple-DES	Data Encryption and Decryption
CTS mode	Ciphertext stealing mode of operation
Generic GCM encryption with external IV (non-compliant with IG A.5)	Data Encryption
RFC4106 GCM encryption with external IV (non-compliant with IG A.5)	Data Encryption
LRW mode	Mode of operation introduced by Liskov, Rivest and Wagner
PCBC mode	Propagating cipher block chaining mode of operation
MD4	Message digest
MD5	Message digest
Poly1305	Message digest
RIPEMD	Message digest
Tiger	Message digest
Whirlpool	Message digest
Michael Mic	Message authentication code
XCBC	Message authentication code
RSA primitive operations	RSA primitive operations including encryption, decryption, sign and verify

Table 10: Non-Approved and not-Allowed 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 11 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			
RSA public key for integrity tests	N/A	N/A	N/A
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	Updated during DRBG initialization	N/A	Zeroized when freeing the cipher handler.

Table 11: 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 creation of random numbers. In addition, the module provides a Random Number Generation service to the calling applications.

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

To seed the DRBG, the module uses a Non-Deterministic Random Number Generator (NDRNG) as the entropy source. The NDRNG is based on the Linux RNG (within the module's physical boundary but outside of its logical boundary) and the CPU-Jitter RNG (with the module's logical boundary). The NDRNG provides sufficient entropy to the DRBG during initialization and reseeding.

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

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/CSP Entry and Output

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

7.4 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 RSA public keys used for

the integrity tests, which are stored in the module and rely on the operating system for protection.

7.5 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 (please see the API document for full details).

7.6 Key Wrapping

The module provides AES key wrapping using KW mode.

- AES key wrapping provides between 128 and 256 bits of encryption strength.

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 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. If the power-up self-tests pass, no specific message is prompted. If the power-up self-tests fail, the module will output an error message and enters the error state. No further operations are possible when the module is in the error state.

On-demand self-tests can be invoked by rebooting the operating system. Table 12 lists all the self-tests performed by the module.

Self Test	Description
Power-up tests performed at power-up and on demand:	
Cryptographic Algorithm Known Answer Tests (KATs)	KATs for AES (XTS, CMAC, CBC, CTR, OFB, GCM, CCM and KW block chaining modes) encryption and decryption are performed separately. KATs for Triple-DES (CBC, CTR and CMAC block chaining modes) 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 and HMAC-SHA-512 are performed. KATs for Hash_DRBG, HMAC_DRBG and CTR_DRBG are performed. KAT for RSA signature verification is performed when kernel object files are loaded at boot-up time and their signatures are verified.
Software Integrity Test	The module uses the HMAC-SHA-256 algorithm for the integrity test of static kernel binary. The HMAC calculation is performed by the fipscheck application, which is part of the module. The HMAC-SHA-256 algorithm is provided by the OpenSSL bound module. RSA signature verification is performed when loading each of the kernel object files during boot-up time provided by the module itself.
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 12: 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
x86_64	kernel-default-4.4.59-92.24.2.x86_64.rpm dracut-fips-044.1-109.17.1.x86_64.rpm
z System	kernel-default-4.4.59-92.24.2.s390x.rpm dracut-fips-044.1-109.17.1.s390x.rpm

Table 13: 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 13 using the zypper tool. The integrity of the RPM package is automatically verified during the installation, and the Crypto Officer shall not install the RPM package if there is any integrity error.

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 or zipl.conf file:

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

Please note on z system, it is not required to pass the boot and fips=1 options to zipl. To prevent zipl from failing to find the correct boot device, edit /boot/zipl/config to remove these values and run zipl -Vnc /boot/zip/config.

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, on x86_64 platform, 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 compoent 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.

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.

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. It is compliant with IG A.5, provision 1 (“IPSec protocol IV generation”).

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^{28} 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
KW	Key Wrap
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**
http://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**
http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf

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

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

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

FIPS198-1 **The Keyed Hash Message Authentication Code (HMAC)**
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**
http://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**
http://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**
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf

SP800-38C **NIST Special Publication 800-38C - Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality**
http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf

SP800-38D **NIST Special Publication 800-38D - Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC**
http://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**
http://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** http://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** http://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** http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf