Kernel Crypto API Cryptographic Module

version 1.0

FIPS 140-2 Non-Proprietary Security Policy

Version 1.2

Last update: 2017-07-18

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of Contents

Copyrights and Trademarks

Ubuntu and Canonical are registered trademarks of Canonical Ltd.

Linux is a registered trademark of Linus Torvalds.

1. Cryptographic Module Specification

This document is the non-proprietary FIPS 140-2 Security Policy for version 1.0 of the Ubuntu Kernel Crypto API Cryptographic Module. It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS PUB 140-2 (Federal Information Processing Standards Publication 140-2) for a Security Level 1 software module.

The following sections describe the cryptographic module and how it conforms to the FIPS 140-2 specification in each of the required areas.

1.1. Module Overview

The Ubuntu Kernel Crypto API Cryptographic Module (hereafter referred to as “the module”) is a software module running as part of the operating system kernel that provides general purpose cryptographic services. The module provides cryptographic services to kernel applications through a C language Application Program Interface (API) and to applications running in the user space through an AF_ALG socket type interface. The module utilizes processor instructions to optimize and increase the performance of cryptographic algorithms.

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. The table below 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
Overall Level		1

Table 1 - Security Levels

The cryptographic logical boundary consists of all kernel objects and the integrity check files used for Integrity Tests. The table below enumerates the components that comprise the module with their location in the target platform.

Description	Components
Integrity test utility	/usr/bin/sha512hmac
Integrity check HMAC file for the integrity test utility.	/usr/bin/.sha512hmac.hmac
Static kernel binary	On Power system: /boot/vmlinux-4.4.0-1002-fips
	On x86_64 and z system: /boot/vmlinuz-4.4.0-1002-fips
Integrity check HMAC file for static kernel binary	On Power system: /boot/.vmlinux-4.4.0-1002-fips.hmac
	On x86_64 and z system: /boot/.vmlinuz-4.4.0-1002-fips.hmac
Cryptographic kernel object files	On Power system: /lib/modules/4.4.0-1002-fips/kernel/crypto/*.ko /lib/modules/4.4.0-1002-fips/kernel/arch/powerpc/crypto/*.ko /lib/modules/4.4.0-1002-fips/kernel/drivers/crypto/vmx/*.ko
	On x86_64 system: /lib/modules/4.4.0-1002-fips/kernel/crypto/*.ko /lib/modules/4.4.0-1002-fips/kernel/arch/x86/crypto/*.ko
	On z system: /lib/modules/4.4.0-1002-fips/kernel/crypto/*.ko /lib/modules/4.4.0-1002-fips/kernel/arch/s390/crypto/*.ko

Table 2 - Cryptographic Module Components

The software block diagram below shows the module, its interfaces with the operational environment and the delimitation of its logical boundary, comprised of all the components within the **BLUE** box.

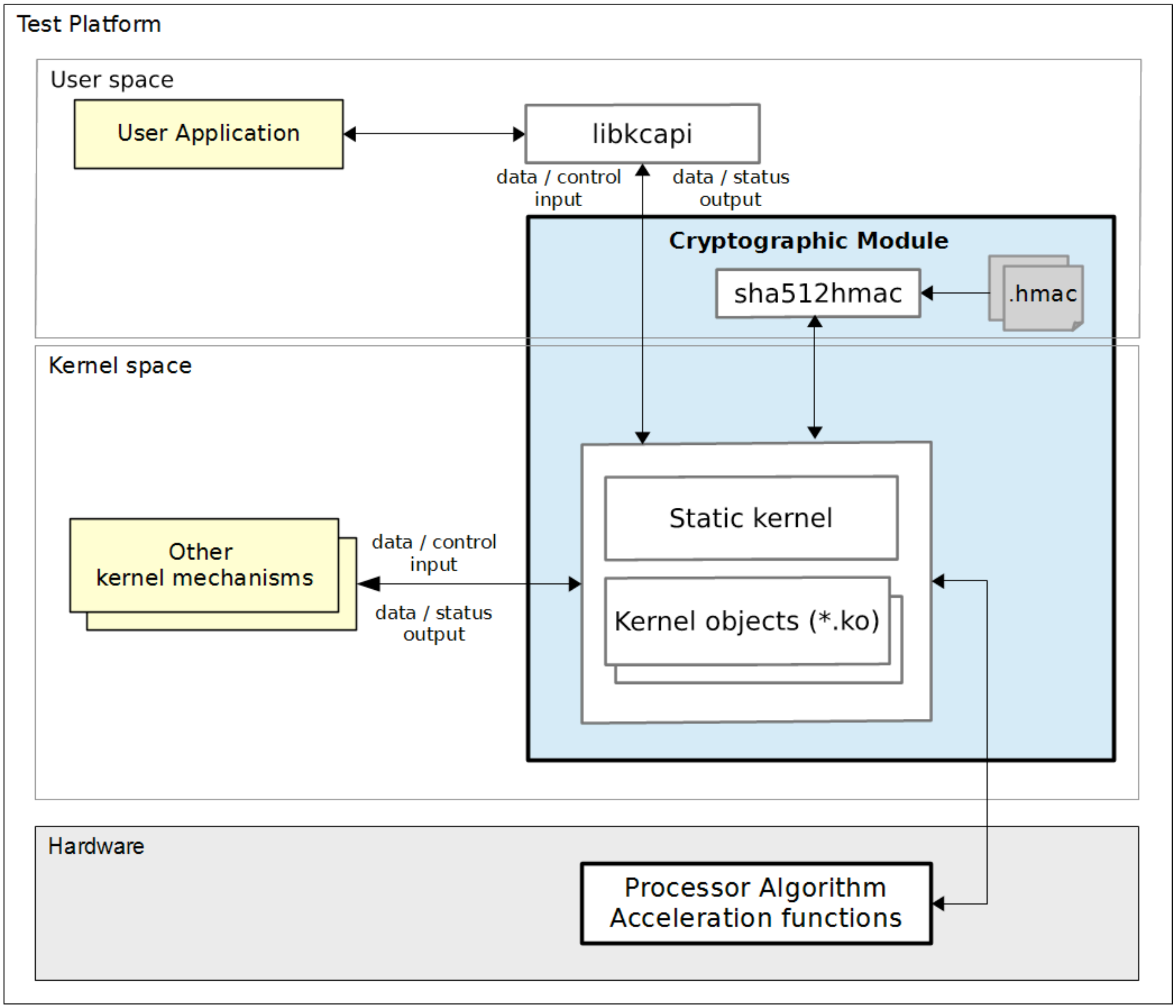

Figure 1 - Software Block Diagram

The module is aimed to run on a general purpose computer (GPC); the physical boundary of the module is the tested platforms. Figure 2 shows the major components of a GPC.

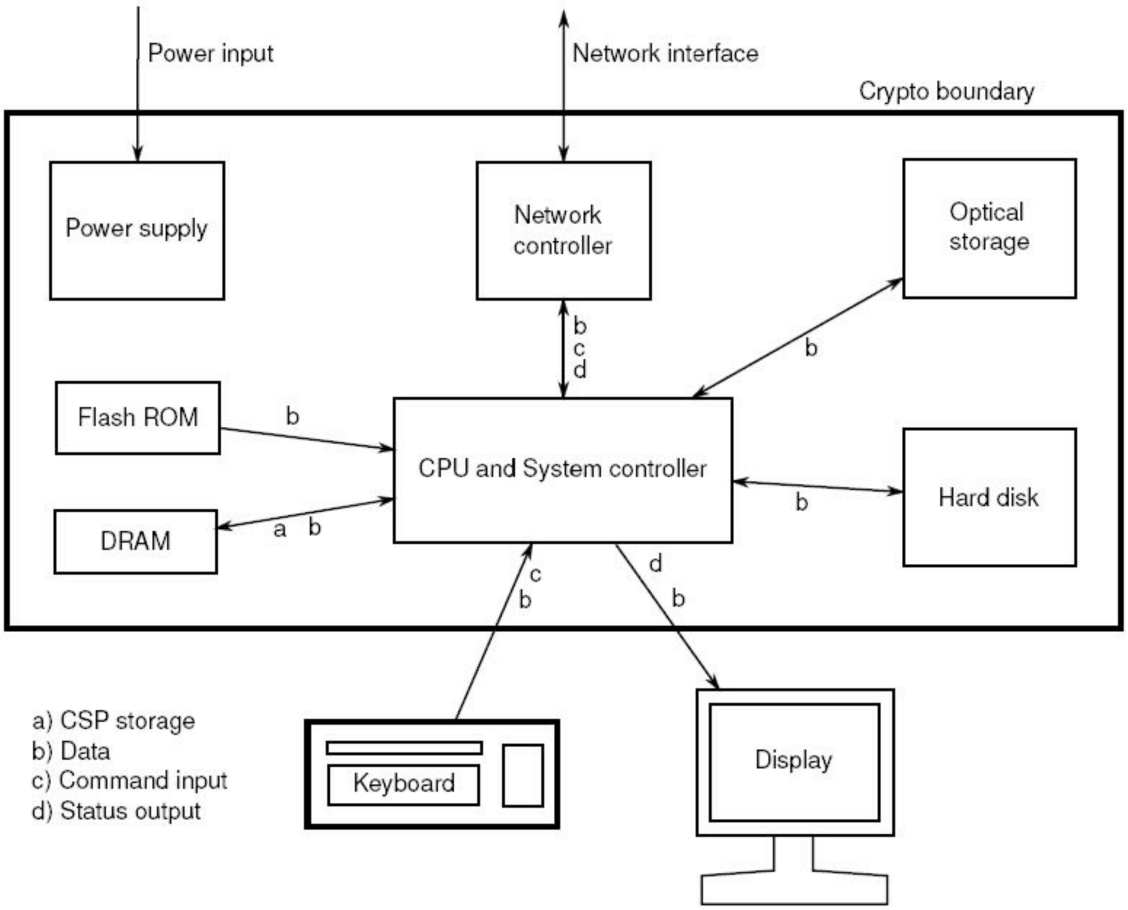

Figure 2 - Cryptographic Module Physical Boundary

The module has been tested on the test platforms shown below.

Test Platform	Processor	Processor Architecture	Test Configuration
IBM Power System S822L (PowerNV 8247-22L)	POWER8	Power system	Ubuntu 16.04 LTS 64-bit Little Endian with/without Power ISA 2.07 (PAA)
IBM Power System S822LC (PowerNV 8001-22C)	POWER8	Power system	Ubuntu 16.04 LTS 64-bit Little Endian with/without Power ISA 2.07 (PAA)
IBM Power System S822LC (PowerNV 8335-GTB)	POWER8	Power system	Ubuntu 16.04 LTS 64-bit Little Endian with/without Power ISA 2.07 (PAA)

Test Platform	Processor	Processor Architecture	Test Configuration
Supermicro SYS-5018R-WR	Intel® Xeon® CPU E5-2620v3	x86_64	Ubuntu 16.04 LTS 64-bit with/without AES-NI (PAA)
IBM z13	z13	z System	Ubuntu 16.04 LTS 64-bit running on LPAR with/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.

1.2. Modes of Operation

The module supports two modes of operation:

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

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

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

2. Cryptographic Module Ports and Interfaces

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

The logical interfaces are the API through which kernel modules request services, and the AF_ALG type socket that allows the applications running in the user space to request cryptographic services from the module. The following table summarizes the four logical interfaces:

FIPS Interface	Physical Port	Logical Interface
Data Input	Keyboard	API input parameters from kernel system calls, AF_ALG type socket.
Data Output	Display	API output parameters from kernel system calls, AF_ALG type socket.
Control Input	Keyboard	API function calls, API input parameters for control from kernel system calls, AF_ALG type socket, kernel command line.
Status Output	Display	API return codes, AF_ALG type socket, kernel logs.
Power Input	PC Power Supply Port	N/A

Table 4 - Ports and Interfaces

3. Roles, Services and Authentication

3.1. Roles

The module supports the following roles:

- **User role**: performs cryptographic services (in both FIPS mode and non-FIPS mode), key zeroization, show status, and on-demand self-test.
- **Crypto Officer role**: performs module installation and initialization.

The User and Crypto Officer roles are implicitly assumed by the entity accessing the module services.

3.2. Services

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

The table below shows the services available in FIPS mode. For each service, the associated cryptographic algorithms, the roles to perform the service, and the cryptographic keys or Critical Security Parameters and their access right are listed. If the services involve the use of the cryptographic algorithms, the corresponding Cryptographic Algorithm Validation System (CAVS) certificate numbers of the cryptographic algorithms can be found in Table 7, Table 8 and Table 9 of this security policy.

Service	Algorithms	Role	Access	Keys/CSP
Cryptographic Library Services				
Symmetric Encryption and Decryption	AES	User	Read	AES key
	Triple-DES	User	Read	Triple-DES key
Random number generation	DRBG	User	Read, Update	Entropy input string, Internal state
Message digest	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	User	N/A	N/A
Message authentication code (MAC)	HMAC	User	Read	HMAC key
	CMAC with AES	User	Read	AES key
	CMAC with Triple-DES	User	Read	Triple-DES key
Key wrapping	AES	User	Read	AES key
Encrypt-then-MAC (authenc) operation for IPsec	AES (CBC mode), Triple-DES (CBC mode), HMAC	User	Read	AES key, Triple-DES key, HMAC key

Service	Algorithms	Role	Access	Keys/CSP
Other Services				
Error detection code	crc32c[1], crct10dif[1]	User	N/A	None
Data compression	deflate[1], lz4[1], lz4hc[1], lzo[1], zlib[1], 842[1]	User	N/A	None
Memory copy operation	ecb(cipher_null)[1]	User	N/A	None
Show status	N/A	User	N/A	None
Zeroization	N/A	User	Zeroize	All CSPs
Self-Tests	AES, Triple-DES, SHS, HMAC, RSA, DRBG	User	N/A	None
Module installation	N/A	Crypto Officer	N/A	None
Module initialization	N/A	Crypto Officer	N/A	None

Table 5 - Services in FIPS mode of operation

The table below lists the services only available in non-FIPS mode of operation.

Service	Algorithms / Key sizes	Role	Access	Keys/CSPs
Symmetric encryption and decryption	Anubis, CAST5, CAST6, DES, Serpent, SEED, Blowfish, Twofish, RC4, FCrypt, Khazad, ChaCha20, Camellia, TEA, XTEA, XETA, Salsa20 listed in Table 12	User	Read	Symmetric key
	2-key Triple-DES listed in Table 12, CMAC with 2-key Triple-DES	User	Read	2-key Triple-DES key
	Generic GCM encryption with external IV, RFC4106 GCM encryption with external IV listed in Table 12	User	Read	AES key
	CTS, PCBC and LRW modes of operation listed in Table 12	User	Read	Symmetric key
Authenticated Encryption	Chacha20-Poly1305 listed in Table 12	User	Read	Symmetric key and MAC key

[1] This algorithm does not provide any cryptographic attribute.

Service	Algorithms / Key sizes	Role	Access	Keys/CSPs
Message digest	MD4, MD5, RIPEMD, Tiger, Whirlpool, Poly1305, GHASH listed in Table 12	User	N/A	none
Message authentication code (MAC)	HMAC with less than 112 bit keys listed in Table 12	User	Read	HMAC key
	CMAC with 2-key Triple-DES	User	Read	2-key Triple-DES key
	Michael Mic, VMAC, XCBC listed in Table 12	User	Read	MAC Key
RSA primitive operations including encryption, decryption, sign and verify	RSA primitive operations listed in Table 12	User	Read	RSA key pair

Table 6 – Services in non-FIPS mode of operation

3.3. Algorithms

The algorithms implemented in the module are tested and validated by CAVP for the following operating environment:

- Ubuntu 16.04 LTS 64-bit Little Endian running on POWER system
- Ubuntu 16.04 LTS 64-bit running on Intel® Xeon® processor
- Ubuntu 16.04 LTS 64-bit running on z system

The Ubuntu Kernel Crypto API Cryptographic Module is compiled to use the support from the processor and assembly code for AES, Triple-DES, SHA and GHASH[1] operations to enhance the performance of the module. Different implementations can be invoked by using the unique algorithm driver names. All the algorithm execution paths have been validated by CAVP.

3.3.1. Ubuntu 16.04 LTS 64-bit Little Endian Running on POWER System

On the platform that runs the POWER system, the module supports the use of generic C implementation for all the algorithms, the use of Power ISA 2.07 for AES core algorithm, the use of Power ISA2.07 for AES (both core and modes) and GHASH, and the use of strict assembler for SHA-1 algorithm.

The following table shows the CAVS certificates and their associated information of the cryptographic implementation in FIPS mode.

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use

[1] The GHASH algorithm is used in GCM mode.

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
Generic C implementation for AES: #4489 Using Power ISA 2.07 for AES core: #4492	AES	[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
RFC4106 GCM with external IV using C implementation for AES: #4491 RFC4106 GCM with external IV using Power ISA 2.07 for AES core: #4494	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
RFC4106 GCM with internal IV using C implementation for AES: #4490 RFC4106 GCM with internal IV using Power ISA 2.07 for AES core: #4493	AES	[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
Using Power ISA 2.07 for AES and	AES	[FIPS197], [SP800-38A]	CBC, CTR	128, 192, 256	Data Encryption and Decryption

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
GHASH: #4495		[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
RFC4106 GCM with external IV using Power ISA 2.07 for AES and GHASH: #4497	AES	[FIPS197], [SP800-38A]	CBC, CTR	128, 192, 256	Data Encryption and Decryption
		[SP800-38D] [RFC4106]	RFC4106 GCM with external IV	128, 192, 256	Data Decryption
RFC4106 GCM with internal IV using Power ISA 2.07 for AES and GHASH: #4496	AES	[FIPS197], [SP800-38A]	CBC, CTR	128, 192, 256	Data Encryption and Decryption
		[SP800-38D] [RFC4106]	RFC4106 GCM with internal IV	128, 192, 256	Data Encryption
Generic C implementation for SHA: #1463 Strict assembler for SHA-1[1]: #1466	DRBG	[SP800-90A]	**Hash_DRBG:** SHA-1[1], SHA-256, SHA-384, SHA-512 with/without PR	N/A	Deterministic Random Bit Generation
			HMAC_DRBG: SHA-1[1], SHA-256, SHA-384, SHA-512 with/without PR		

[1] Only SHA-1 is supported and tested using strict assembler for SHA-1 implementation.

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
Generic C implementation for AES: #1463 Using Power ISA 2.07 for AES core: #1464 Using Power ISA 2.07 for AES: #1465			**CTR_DRBG:** AES-128, AES-192, AES-256 with DF, with/without PR		
Generic C implementation for SHA: #2974 Strict assembler for SHA-1[1]: #2975	HMAC	[FIPS198-1]	SHA-1[1], SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message authentication code
Generic C implementation for SHA: #2451 Strict assembler for SHA-1[1]: #2452	RSA	[FIPS186-4]	**PKCS#1v1.5** SHA-1[1], SHA-224, SHA-256, SHA-384, SHA-512	1024 or greater	Digital Signature Verification for integrity tests of kernel object files.
Generic C implementation for SHA: #3691 Strict assembler for SHA-1[1] #3692	SHS	[FIPS180-4]	SHA-1[1], SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest
Generic C implementation	Triple-DES	[SP800-67], [SP800-38A]	ECB, CBC, CTR	192	Data Encryption and Decryption

[1] Only SHA-1 is supported and tested using strict assembler for SHA-1 implementation.

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
for Triple-DES: #2404		[SP800-67], [SP800-38B]	CMAC	192	MAC Generation and Verification

Table 7 – Cryptographic Algorithms for POWER system

3.3.2. Ubuntu 16.04 LTS 64-bit Running on Intel® Xeon® Processor

On the platform that runs the Intel Xeon processor, the module supports the use of generic C implementation for all the algorithms, the use of strict assembler for AES and Triple-DES core algorithms, the use of strict assembler for Triple-DES (both core and modes), the use of AES-NI for AES core algorithm and CLMUL for the GHASH algorithm, the use of AES-NI for AES (both core and modes), the use of AVX, AVX2 and SSSE3 for SHA algorithm, and the use of multi-buffer for SHA-1 algorithm.

The following table shows the CAVS certificates and their associated information of the cryptographic implementation in FIPS mode.

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
Generic C implementation for AES: #4478 Strict assembler for AES core: #4481 Using AES-NI for AES core and CLMUL for GHASH: #4484	AES	[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
RFC4106 GCM with external IV	AES	[FIPS197], [SP800-38A]	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
using C implementation for AES: #4480 RFC4106 GCM with external IV using strict assembler for AES core: #4483 RFC4106 GCM with external IV using AES-NI for AES core and CLMUL for GHASH: #4486		[SP800-38D] [RFC4106]	RFC4106 GCM with external IV	128, 192, 256	Data Decryption
RFC4106 GCM with internal IV using C implementation for AES: #4479 RFC4106 GCM with internal IV using strict assembler for AES core: #4482 RFC4106 GCM with internal IV using AES-NI for AES core and CLMUL for GHASH: #4485	AES	[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
Using AES-NI for AES and RFC4106 GCM with external IV:	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

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
#4487		[SP800-38E]	XTS	128, 256	Data Encryption and Decryption for Data Storage
Using AES-NI for AES and RFC4106 GCM with internal IV: #4488	AES	[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
Generic C implementation for SHA: #1457 Using AVX for SHA: #1460 Using AVX2 for SHA: #1461 Using SSSE3 for SHA: #1462	DRBG	[SP800-90A]	**Hash_DRBG:** SHA-1, SHA-256, SHA-384, SHA-512 with/without PR	N/A	Deterministic Random Bit Generation
			HMAC_DRBG: SHA-1, SHA-256, SHA-384, SHA-512 with/without PR		
Generic C implementation for AES: #1457 Strict assembler for AES core: #1458 Using AES-NI for AES core: #1459			**CTR_DRBG:** AES-128, AES-192, AES-256 with DF, with/without PR		

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
Generic C implementation for SHA: #2970 Using AVX for SHA: #2971 Using AVX2 for SHA: #2972 Using SSSE3 for SHA: #2973	HMAC	[FIPS198-1]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message authentication code
Generic C implementation for SHA: #2447 Using AVX for SHA: #2448 Using AVX2 for SHA: #2449 Using SSSE3 for SHA: #2450	RSA	[FIPS186-4]	**PKCS#1v1.5** SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024 or greater	Digital Signature Verification for integrity tests.

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
Generic C implementation for SHA: #3687 Using AVX for SHA: #3688 Using AVX2 for SHA: #3689 Using SSSE3 for SHA: #3690 Using multi-buffer for SHA-1[1]: #3695	SHS	[FIPS180-4]	SHA-1[1], SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest
Generic C implementation for Triple-DES: #2401 Strict assembler for Triple-DES core: #2402	Triple-DES	[SP800-67], [SP800-38A]	ECB, CBC, CTR	192	Data Encryption and Decryption
		[SP800-67], [SP800-38B]	CMAC	192	MAC Generation and Verification
Strict assembler for Triple-DES: #2403	Triple-DES	[SP800-67], [SP800-38A]	ECB, CBC, CTR	192	Data Encryption and Decryption

Table 8 – Cryptographic Algorithms for Intel® Xeon® Processor

3.3.3. Ubuntu 16.04 LTS 64-bit Running on z System

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. If CPACF is available in the operational environment, the module uses the support from CPACF automatically. Otherwise, the module uses the C implementation of the algorithms.

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

The following table shows the CAVS certificates and their associated information of the cryptographic implementation in FIPS mode.

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
Generic C implementation for AES: #4498 Using CPACF for AES core: #4502	AES	[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
RFC4106 GCM with external IV using C implementation for AES: #4500 RFC4106 GCM with external IV using CPACF for AES core: #4504 RFC4106 GCM with external IV using CPACF for AES and GHASH: #4507	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
RFC4106 GCM with internal IV	AES	[FIPS197], [SP800-38A]	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
using C implementation for AES: #4501 RFC4106 GCM with internal IV using CPACF for AES core: #4503 RFC4106 GCM with internal IV using CPACF for AES and GHASH: #4506		[SP800-38D] [RFC4106]	RFC4106 GCM with internal IV	128, 192, 256	Data Encryption
Using CPACF for AES and GHASH: #4505	AES	[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]	Generic GCM with external IV	128, 192, 256	Data Encryption and Decryption
		[SP800-38E]	XTS	128, 256	Data Encryption and Decryption for Data Storage
Generic C implementation for SHA: #1467 Using CPACF for SHA: #1469	DRBG	[SP800-90A]	**Hash_DRBG:** SHA-1, SHA-256, SHA-384, SHA-512 with/without PR	N/A	Deterministic Random Bit Generation
			HMAC_DRBG: SHA-1, SHA-256, SHA-384, SHA-512 with/without PR		

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
Generic C implementation for AES: #1467 Using CPACF for AES core: #1469 Using CPACF for AES: #1470			**CTR_DRBG:** AES-128, AES-192, AES-256 with DF, with/without PR		
Generic C implementation for SHA: #2976 Using CPACF for SHA: #2977	HMAC	[FIPS198-1]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message authentication code
Generic C implementation for SHA: #2453 Using CPACF for SHA: #2454	RSA	[FIPS186-4]	**PKCS#1v1.5** SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024 or greater	Digital Signature Verification for integrity tests.
Generic C implementation for SHA: #3693 Using CPACF for SHA: #3694	SHS	[FIPS180-4]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest
Generic C implementation for Triple-DES: #2405 Using CPACF for Triple-DES core: #2406	Triple-DES	[SP800-67], [SP800-38A]	ECB, CBC, CTR	192	Data Encryption and Decryption
		[SP800-67], [SP800-38B]	CMAC	192	MAC Generation and Verification

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
Using CPACF for Triple-DES: #2407	Triple-DES	[SP800-67], [SP800-38A]	ECB, CBC, CTR	192	Data Encryption and Decryption

Table 9 – Cryptographic Algorithms for z system

The CPACF provided by the IBM z system contains the complete AES, Triple-DES and SHA implementations. The following table shows the CAVS certificates and their associated information of the algorithms tested directly from the CPACF:

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
#3958	AES	[FIPS197], [SP800-38A]	ECB, CBC, CTR	128, 192, 256	Data Encryption and Decryption
#3196	SHS	[FIPS180-4]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest
#2214	Triple-DES	[SP800-67], [SP800-38A]	ECB, CBC	192	Data Encryption and Decryption

Table 10 – Cryptographic Algorithms from CPACF

3.3.4. Non-Approved Algorithms

The following table describes the non-Approved but allowed algorithms in FIPS mode:

Algorithm	Use
NDRNG (based on Linux RNG and CPU-Jitter RNG)	The module obtains the entropy data from NDRNG to seed the DRBG.

Table 11 – FIPS-Allowed Cryptographic Algorithms

The table below shows the non-Approved cryptographic algorithms implemented in the module that are only available in non-FIPS mode.

Algorithm	Implementation Name	Use
Anubis	"anubis"	Data Encryption / Decryption
CAST5	"cast5"	Data Encryption / Decryption
CAST6	"cast6"	Data Encryption / Decryption

DES	"des"	Data Encryption / Decryption
Serpent	"serpent" or "tnepres"	Data Encryption / Decryption
SEED	"seed"	Data Encryption / Decryption
Blowfish	"blowfish"	Data Encryption / Decryption
Twofish	"twofish"	Data Encryption / Decryption
RC4	"arc4"	Data Encryption / Decryption
FCrypt	"fcrypt"	Data Encryption / Decryption
Khazad	"khazad"	Data Encryption / Decryption
ChaCha20	"chacha20"	Data Encryption / Decryption
Camellia	"camellia"	Data Encryption/ Decryption
TEA	"tea"	Data Encryption/ Decryption
XTEA	"xtea"	Data Encryption/ Decryption
XETA	"xeta"	Data Encryption/ Decryption
Salsa20	"salsa20"	Data Encryption/ Decryption
2-key Triple-DES	"des3_ede"	Data Encryption / Decryption
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
CTS	"cts"	Ciphertext stealing mode of operation
PCBC	"pcbc"	Propagating Cipher Block Chaining mode of operation
LRW	"lrw"	Mode of operation introduced by Liskov, Rivest, and Wagner
ChaCha20-Poly1305	"rfc7539esp(chacha20)" or "rfc7539(chacha20)"	Authenticated Encryption
MD4	"md4"	Hashing
MD5	"md5"	Hashing
RIPEMD	"rmd128", "rmd160", "rmd256", "rmd320"	Hashing
Tiger	"tgr128", "tgr160", "tgr192"	Hashing
Whirlpool	"wp256", "wp384", "wp512"	Hashing
Poly1305	"poly1305"	Hashing

GHASH	"ghash"	Hashing
HMAC with less than 112 bits key	"hmac"	Message Authentication Code
Michael Mic	"michael_mic"	Message Authentication Code
VMAC	"vmac"	Message Authentication Code
XCBC	"xcbc"	Message Authentication Code
RSA primitive operations	"rsa"	RSA primitive operations including Encryption, Decryption, Sign, Verify

Table 12 - Non-Approved Cryptographic Algorithms and Modes

Note: Calling any algorithm, mode or combination using any of the above listed non-Approved items will cause the module to enter non-FIPS mode implicitly.

3.4. Operator Authentication

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

4. Physical Security

The module is comprised of software only and therefore this security policy does not make any claims on physical security.

5. Operational Environment

5.1. Applicability

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

5.2. Policy

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

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

6. Cryptographic Key Management

The following table summarizes the Critical Security Parameters (CSPs) and public keys that are used by the cryptographic services implemented in the module:

Name	Generation	Entry and Output	Zeroization

AES keys	N/A	The key is passed into the module via API input parameters in plaintext.	`crypto_free_cipher()`
			`crypto_free_ablkcipher()`
			`crypto_free_blkcipher()`
			`crypto_free_skcipher()`
			`crypto_free_aead()`

Triple-DES keys			
HMAC keys	N/A	The key is passed into the module via API input parameters in plaintext.	`crypto_free_shash()` `crypto_free_ahash()`
RSA public key (used for integrity tests)	N/A	N/A	N/A
Entropy input string	Obtained from NDRNG.	None	`crypto_free_rng()`
DRBG internal state (V, C, Key)	During DRBG initialization.	None	`crypto_free_rng()`

Table 13 - Life cycle of Critical Security Parameters (CSP) and public keys

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

6.1. Random Number Generation

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

The DRBG supports the Hash_DRBG, HMAC_DRBG and CTR_DRBG mechanisms. The DRBG is initialized during module initialization; the module loads by default the DRBG using the HMAC_DRBG mechanism with SHA-256 without prediction resistance.

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 (within the module's logical boundary). The NDRNG provides sufficient entropy to the DRBG during initialization (seed) and reseeding (reseed).

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].

6.2. Key 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 which can be used as key material for symmetric algorithms or HMAC.

6.3. Key Agreement / Key Transport / Key Derivation

The module provides key wrapping using the AES with KW mode.

According to Table 2: Comparable strengths in [SP 800-57], the key sizes of AES provides the following security strength in FIPS mode of operation:

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

6.4. Key Entry / Output

The module does not support manual key entry. The keys are provided to the module via API input parameters in plaintext form. This is allowed by [FIPS140-2_IG] IG 7.7, according to the "CM Software to/from App Software via GPC INT Path" entry on the Key Establishment Table.

6.5. Key / CSP Storage

Symmetric keys and HMAC keys are provided to the module by the calling application via API input parameters, and are destroyed by the module when invoking the appropriate API function calls.

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

6.6. Key / CSP Zeroization

The memory occupied by keys is allocated by regular memory allocation operating system calls. Memory is automatically overwritten with "zeroes" and deallocated when the cipher handler is freed.

7. Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The test platforms listed in Table 3 - Tested Platforms have been tested and found to conform to the EMI/EMC requirements specified by 47 Code of Federal Regulations, FCC PART 15, Subpart B, Unintentional Radiators, Digital Devices, Class A (i.e., Business use). These devices are designed to provide reasonable protection against harmful interference when the devices are operated in a commercial environment. They shall be installed and used in accordance with the instruction manual.

8. Self-Tests

FIPS 140-2 requires that the module perform power-up tests to ensure the integrity of the module and the correctness of the cryptographic functionality at start up. In addition, the module performs conditional test for NDRNG. If any self-test fails, the kernel panics and the module enters the error state. In error state, no data output or cryptographic operations are allowed. See section 9.2.3 for details to recover from the error state.

8.1. Power-Up Tests

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

While the module is executing the power-up tests, services are not available, and input and output are inhibited. The module will not return the control to the calling application until the power-up tests are completed successfully.

8.1.1. Integrity Tests

The module verifies its integrity through the following mechanisms:

- All kernel object (*.ko) files are signed with a 4096-bit RSA private key and SHA-512. Before these kernel objects are loaded into memory, the module performs RSA signature verification by using the RSA public key from the X.509 certificates that are compiled into the module's binary. If the signature cannot be verified, the kernel panics to indicate that the test fails and the module enters the error state.
- The integrity of the static kernel binary (i.e. /boot/vmlinux-4.4.0-1002-fips or /boot/vmlinuz-4.4.0-1002-fips file) is ensured with the HMAC-SHA-512 value stored in the .hmac file (i.e. /boot/.vmlinux-4.4.0-1002-fips.hmac or /boot/.vmlinuz-4.4.0-1002-fips.hmac file) that was computed at build time. At run time, the module invokes the sha512hmac utility to calculate the HMAC value of the static kernel binary file, and then compares it with the pre-stored one. If the two HMAC values do not match, the kernel panics to indicate that the test fails and the module enters the error state.
- The Integrity of the sha512hmac utility (i.e. /usr/bin/sha512hmac) is ensured with the HMAC-SHA-512 value stored in the .hmac file (i.e. /usr/bin/.sha512hmac.hmac) that was computed at build time. At run time, the utility itself calculates the HMAC value of the utility, and then compares it with the pre-stored one. If the two HMAC values do not match, the kernel panics to indicate that the test fails and the module enters the error state.

Both the RSA signature verification and HMAC-SHA-512 algorithms are approved algorithms implemented in the module.

8.1.2. Cryptographic Algorithm Tests

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

Algorithm	Power-Up Tests
AES	• KAT of AES in ECB mode with 128, 192 and 256 bit keys, encryption • KAT of AES in ECB mode with 128, 192 and 256 bit keys, decryption • KAT of AES in CBC mode with 128, 192 and 256 bit keys, encryption • KAT of AES in CBC mode with 128, 192 and 256 bit keys, decryption • KAT of AES in CTR mode with 128, 192 and 256 bit keys, encryption • KAT of AES in CTR mode with 128, 192 and 256 bit keys, decryption • KAT of AES in GCM mode with 128, 192 and 256 bit keys, encryption • KAT of AES in GCM mode with 128, 192 and 256 bit keys, decryption • KAT of AES in CCM mode with 128 bit key, encryption • KAT of AES in CCM mode with 128 bit key, decryption • KAT of AES in KW mode with 128 bit key, encryption • KAT of AES in KW mode with 256 bit key, decryption • KAT of AES in XTS mode with 128 and 256 bit keys, encryption • KAT of AES in XTS mode with 128 and 256 bit keys, decryption • KAT of AES in CMAC mode with 128 and 256 bit keys
Triple DES	• KAT of 3-key Triple-DES in ECB mode, encryption • KAT of 3-key Triple-DES in ECB mode, decryption • KAT of 3-key Triple-DES in CBC mode, encryption • KAT of 3-key Triple-DES in CBC mode, decryption • KAT of 3-key Triple-DES in CTR mode, encryption • KAT of 3-key Triple-DES in CTR mode, decryption • KAT of 3-key Triple-DES in CMAC mode
SHS	• KAT of SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512
HMAC	• KAT of HMAC-SHA-1 • KAT of HMAC-SHA-224 • KAT of HMAC-SHA-256 • KAT of HMAC-SHA-384 • KAT of HMAC-SHA-512
DRBG	• KAT of Hash_DRBG with SHA-256, with and without PR • KAT of HMAC_DRBG with SHA-256, with and without PR • KAT of CTR_DRBG with AES-128, with and without PR • KAT of CTR_DRBG with AES-192, without PR • KAT of CTR_DRBG with AES-256, without PR
RSA	• KAT of RSA signature verification is covered by the integrity tests which is allowed by [FIPS140-2_IG] IG 9.3

Table 14- Self-Tests

For the KAT, the module calculates the result and compares it with the known value. If the answer does not match the known answer, the KAT is failed and the module enters the Error state.

The KATs cover the different cryptographic implementations available in the operating environment.

8.2. On-Demand Self-Tests

On-Demand self-tests can be invoked by power cycling the module or rebooting the operating system. During the execution of the on-demand self-tests, services are not available and no data output or input is possible.

8.3. Conditional Tests

The module performs the Continuous Random Number Generator Test (CRNGT) shown in the following table:

Algorithm	Conditional Test
NDRNG	• CRNGT

Table 15 - Conditional Tests

9. Guidance

9.1. Crypto Officer Guidance

The binaries of the module are contained in the Debian 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 Debian packages are used to install the FIPS validated module:

Processor Architecture	Debian packages
x86_64	fips-initramfs_0.0.3_amd64.deb linux-fips_4.4.0.1002.3_amd64.deb
Power system	fips-initramfs_0.0.3_ppc64el.deb linux-fips_4.4.0.1002.3_ppc64el.deb
z System	fips-initramfs_0.0.3_s390x.deb linux-fips_4.4.0.1002.3_s390x.deb

Table 16 – Debian packages

9.1.1. Module Installation

The Crypto Officer can install the Debian packages containing the module listed in Table 16 using a normal packaging tool such as Advanced Package Tool (APT). All the Debian packages are associated with hashes for integrity check. The integrity of the Debian package is automatically verified by the packaging tool during the installation of the module. The Crypto Officer shall not install the Debian package if the integrity of the Debian package fails.

To download the FIPS validated version of the module, please contact a Canonical representative for the repository path. Please note that when the FIPS validated version of the module is installed, any custom or Hardware Enablement (HWE) kernel cannot be installed.

9.1.2. Operating Environment Configuration

To configure the operating environment to support FIPS, the following shall be performed with root privileges:

(1) Add `fips=1` to the kernel command line.

- For x86_64 and Power systems, create the file `/etc/default/grub.d/99-fips.cfg` with the content: `GRUB_CMDLINE_LINUX_DEFAULT=”$GRUB_CMDLINE_LINUX_DEFAULT fips=1”`.
- For z system, edit `/etc/zipl.conf` file and append the "`fips=1`" in the parameters line for the specified boot image.

(2) If /boot resides on a separate partition, the kernel parameter `bootdev=UUID=<UUID of partition>` must also be appended in the aforementioned grub or zipl.conf file. Please see the following **Note** for more details.

(3) Update the boot loader.

- For x86_64 and Power systems, execute the `update-grub` command.

- For z system, execute the `zipl` command.

(4) Execute the `reboot` command to reboot the system with the new settings.

The operating environment is now configured to support FIPS operation. The Crypto Officer should check the existence of the file, `/proc/sys/crypto/fips_enabled`, and that it contains “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.

Note: If /boot resides on a separate partition, the kernel parameter `bootdev=UUID=<UUID of partition>` must be supplied. The partition can be identified with the `df /boot` command. For example:

```
$ df /boot
Filesystem     1K-blocks   Used Available Use% Mounted on
/dev/sdb2         241965 127948    101525  56% /boot
```

The UUID of the /boot partition can be found by using the `grep /boot /etc/fstab` command. For example:

```
$ grep /boot /etc/fstab
# /boot was on /dev/sdb2 during installation
UUID=cec0abe7-14a6-4e72-83ba-b912468bbb38 /boot    ext2    defaults     0     2
```

Then, the UUID shall be added in the `/etc/default/grub`. For example:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet bootdev=UUID=cec0abe7-14a6-4e72-83ba-b912468bbb38
fips=1"
```

9.2. User Guidance

For detailed description of the Linux Kernel Crypto API, please refer to the user documentation [KC API Architecture].

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

9.2.1. 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 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 [FIPS140-2_IG] 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 therefore there is no restriction on the IV generation.

9.2.2. AES XTS

As specified in [SP800-38E], the AES algorithm in XTS mode was 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 schema). For dm-crypt, the length of a single data unit encrypted with the XTS-AES 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 [FIPS140-2_IG] IG A.9, the module implements a check to ensure that the two AES keys used in XTS-AES algorithm are not identical.

9.2.3. 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 [FIPS140-2_IG] IG A.13.

9.2.4. Handling FIPS Related Errors

When the module fails any self-test, it will panic the kernel and the operating system will not load. Errors occurred during the self-tests transition the module into the error state. The only way to recover from this error state is to reboot the system. If the failure persists, the module must be reinstalled by the Crypto Officer following the instructions as specified in section 9.1.

The kernel dumps self-test success and failure messages into the kernel message ring buffer. The user can use **dmesg** to read the contents of the kernel ring buffer. The format of the ring buffer (dmesg) output for self-test status 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.

10. Mitigation of Other Attacks

The module does not implement mitigation of other attacks.

Appendix A. Glossary and Abbreviations

AES	Advanced Encryption Standard
AES-NI	Advanced Encryption Standard New Instructions
API	Application Program Interface
APT	Advanced Package Tool
CAVP	Cryptographic Algorithm Validation Program
CAVS	Cryptographic Algorithm Validation System
CBC	Cipher Block Chaining
CCM	Counter with Cipher Block Chaining-Message Authentication Code
CLMUL	Carry-less Multiplication
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CPACF	CP Assist for Cryptographic Function
CRNGT	Continuous Random Number Generator Test
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DF	Derivation Function
DSA	Digital Signature Algorithm
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
EMI/EMC	Electromagnetic Interference/Electromagnetic Compatibility
FCC	Federal Communications Commission
FIPS	Federal Information Processing Standards Publication
GCM	Galois Counter Mode
GPC	General Purpose Computer
HMAC	Hash Message Authentication Code
IG	Implementation Guidance
KAT	Known Answer Test
KDF	Key Derivation Function
KW	Key Wrap
LPAR	Logical Partitions
MAC	Message Authentication Code
NIST	National Institute of Science and Technology

NDRNG	Non-Deterministic Random Number Generator
PAA	Processor Algorithm Acceleration
PAI	Processor Algorithm Implementation
PCT	Pair-wise Consistency Test
PR	Prediction Resistance
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SSSE3	Supplemental Streaming SIMD Extensions 3
XTS	XEX-based Tweaked-codebook mode with ciphertext Stealing

Appendix B. References

FIPS140-2	**FIPS PUB 140-2 - Security Requirements For Cryptographic Modules** May 2001 http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
FIPS140-2_IG	**Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program** June 17, 2016 http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf
FIPS180-4	**Secure Hash Standard (SHS)** March 2012 http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.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
KC API Architecture	**Kernel Crypto API Architecture** 2016 http://www.chronox.de/crypto-API/crypto/architecture.html
PKCS#1	**Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1** February 2003 http://www.ietf.org/rfc/rfc3447.txt
RFC4106	**The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)** June 2005 https://tools.ietf.org/html/rfc4106
RFC6071	**IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap** February 2011 https://tools.ietf.org/html/rfc6071
RFC7296	**Internet Key Exchange Protocol Version 2 (IKEv2)** October 2014 https://tools.ietf.org/html/rfc7296

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://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf
SP800-38D	**NIST Special Publication 800-38D - Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC** November 2007 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
SP800-38F	**NIST Special Publication 800-38F - Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping** December 2012 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** January 2012 http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf
SP800-90A	**NIST Special Publication 800-90A - Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** June 2015 http://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** November 2015 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf