ubuntu 16.04 Kernel Crypto API Cryptographic Module

version 2.0

FIPS 140-2 Non-Proprietary Security Policy

Version 2.4

Last update: 2022-06-23

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 2.0 of the Ubuntu 16.04 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 16.04 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 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	/boot/vmlinuz-4.4.0.1017-fips
Integrity check HMAC file for static kernel binary	/boot/.vmlinuz-4.4.0.1017-fips.hmac
Cryptographic kernel object files	/lib/modules/4.4.0.1017-fips/kernel/crypto/*.ko /lib/modules/4.4.0.1017-fips/kernel/arch/x86/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.

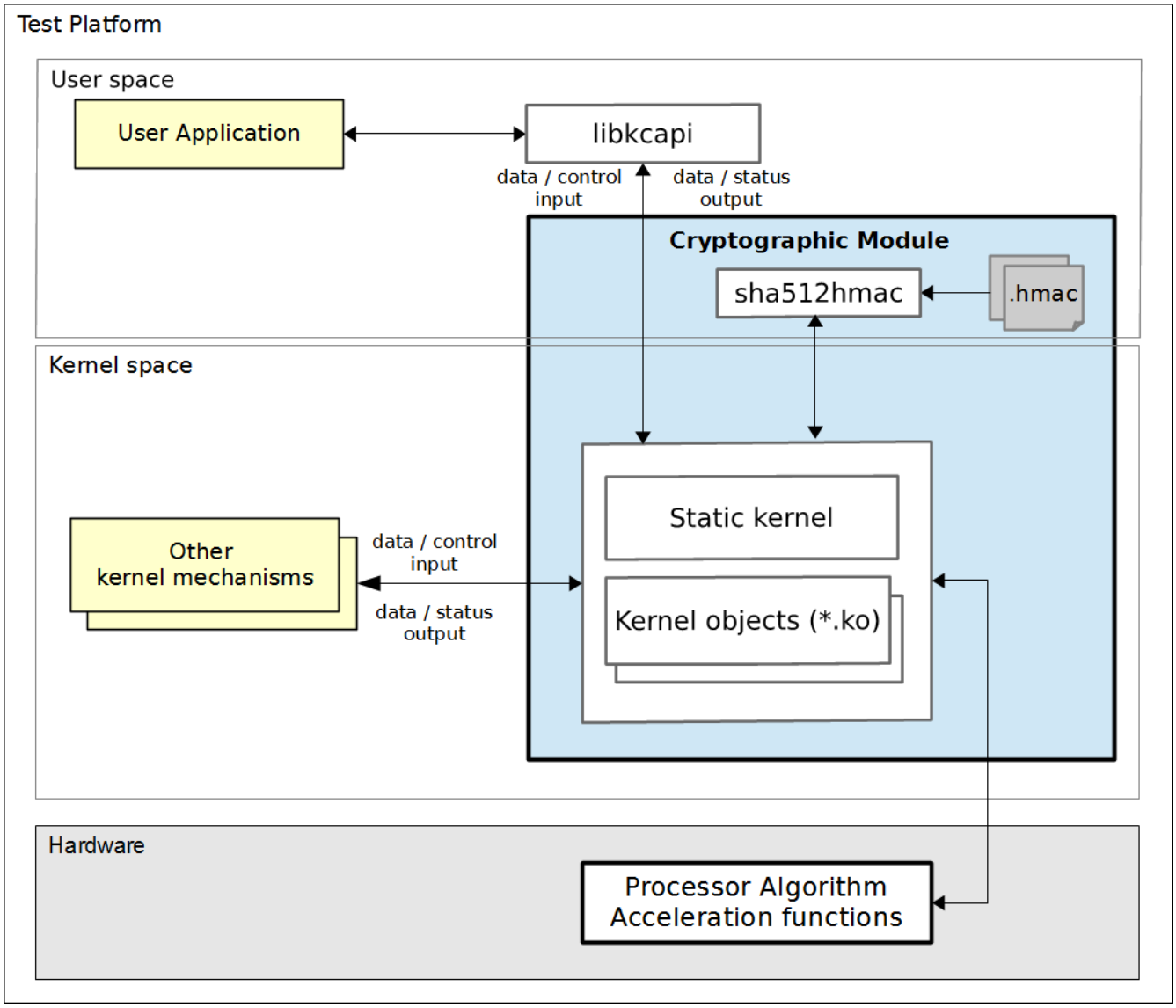

Figure 1 - Software Block Diagram

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.

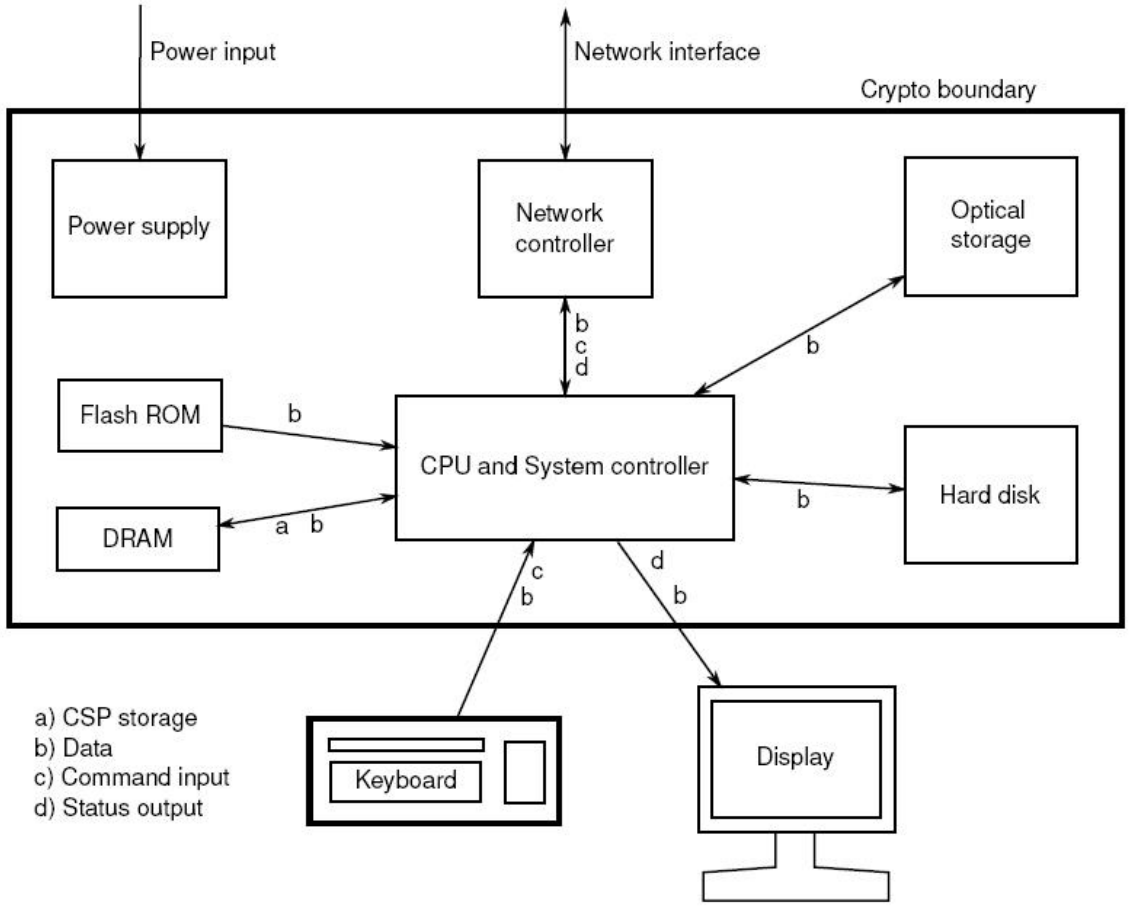

Figure 2 - Cryptographic Module Physical Boundary

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

Test Platform	Processor	Processor Architecture	Test Configuration
Supermicro SYS-5018R-WR	Intel® Xeon® E5	Intel x86 64 bits	Ubuntu 16.04 LTS 64-bit with/without AES-NI (PAA)

Table 3 - Tested Platforms

Note: Per [FIPS 140-2_IG] G.5, the Cryptographic Module Validation Program (CMVP) makes no statement as to the correct operation of the module or the security strengths of the generated keys when this module is ported and executed in an operational environment not listed on the validation certificate.

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 to be 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 platforms 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	GPC 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. The following convention is used to specify access rights to a CSP:

- **Create**: the calling application can create a new CSP.
- **Read**: the calling application can read the CSP.
- **Update**: the calling application can write a new value to the CSP.
- **Zeroize**: the calling application can zeroize the CSP.
- **n/a**: the calling application does not access any CSP or key during its operation.

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.

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	HMAC	User	Read	HMAC key

Service	Algorithms	Role	Access	Keys/CSP
authentication code (MAC)	CMAC with AES	User	Read	AES key
	CMAC with Triple-DES	User	Read	Triple-DES key
Key wrapping (KTS[1])	AES	User	Read	AES key
Encrypt-then-MAC (authenc) operation for IPsec	AES (CBC mode) Triple-DES (CBC mode) HMAC-SHA1/224/256/384/512	User	Read	AES key, Triple-DES key, HMAC key
RSA signature verification	RSA	User	Read	RSA public key
RSA key encapsulation[2]	RSA	User	Read	RSA key pair
Other Services				
Error detection code	crc32c[3], crct10dif[3]	User	N/A	None
Data compression	deflate[3], lz4[3], lz4hc[3], lzo[3], zlib[3]	User	N/A	None
Memory copy operation	ecb(cipher_null)[3] hmac(digest_null)[3]	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.

[1] Approved per IG D.9

[2] Allowed per IG D.9

[3] These algorithms do not provide any cryptographic attribute.

Service	Algorithms / Key sizes	Role	Access	Keys
Symmetric encryption and decryption	AES-XTS with 192-bit key size	User	Read	Symmetric key
	2-key Triple-DES	User	Read	2-key Triple-DES key
	• Generic AES GCM encryption with external IV • RFC4106 AES GCM encryption with external IV	User	Read	AES key
Message digest	• GHASH outside the GCM context • SHA1 using sha1-mb (multi-buffer) implementations	User	N/A	None
Message authentication code (MAC)	HMAC with less than 112 bit keys	User	Read	HMAC key
	CMAC with 2-key Triple-DES	User	Read	2-key Triple-DES key
RSA sign primitive operation		User	Read	RSA private key
ECDSA Key Generation	P-256	User	Read	ECDSA private key
EC Diffie-Hellman shared secret computation	ECDH	User	Read	ECDSA P-256 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 running on Intel® Xeon® E5 processor

The Ubuntu 16.04 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 Running on Intel® Xeon® E5 Processor

On the platform that runs the Intel Xeon E5 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.

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

[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: #C1222 Strict assembler for AES core: #C1220 Using AES-NI for AES core and CLMUL for GHASH: #C1221	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
		[SP800-38D]	GMAC	128, 192, 256	MAC Generation and Verification
		[SP800-38E]	XTS	128, 256	Data Encryption and Decryption for Data Storage
		[SP800-38F]	KW	128, 192, 256	Key Wrapping and Unwrapping
Generic C implementation for AES: #C1214 Strict assembler for AES core: #C1216 Using AES-NI: #C1210 Using AES-NI for AES core and CLMUL for GHASH: #C1212	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 and Decryption
Generic C implementation	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
for AES: #C1213 Strict assembler for AES core: #C1215 Using AES-NI: #C1211 Using AES-NI for AES core and CLMUL for GHASH: #C1303		[SP800-38D] [RFC4106]	RFC4106 GCM with external IV	128, 192, 256	Data Decryption
Generic C implementation for SHA: #C1222 Using AVX for SHA: #C1217 Using AVX2 for SHA: #C1218 Using SSSE3 for SHA: #C1219	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: #C1222 Strict assembler for AES core: #C1220 Using AES-NI for AES core: #C1221			**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: #C1222 Using AVX for SHA: #C1217 Using AVX2 for SHA: #C1218 Using SSSE3 for SHA: #C1219	HMAC	[FIPS198-1]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message authentication code
Generic C implementation for SHA: #C1222 Using AVX for SHA: #C1217 Using AVX2 for SHA: #C1218 Using SSSE3 for SHA: #C1219	RSA	[FIPS186-4]	**PKCS#1v1.5** SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024 up to 4096	Digital Signature Verification for integrity tests.
Generic C implementation for SHA: #C1222 Using AVX for SHA: #C1217 Using AVX2 for SHA: #C1218 Using SSSE3 for SHA: #C1219	SHS	[FIPS180-4]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest
Generic C implementation for Triple-DES: #C1222 Strict assembler for Triple-DES core: #C1220	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
AES-GCM: #C1210, #C1211, #C1212, #C1213, #C1214, #C1215, #C1216, #C1220, #C1221, #C1222, #C1303 (any GCM implementation) AES-CCM: #C1220, #C1221, #C1222 (any CCM implementation) AES-KW: #C1220, #C1221, #C1222 (any KW implementation) AES: #C1210, #C1211, #C1212, #C1213, #C1214, #C1215, #C1216, #C1220, #C1221, #C1222, #C1303 Triple-DES: #C1220, #C1222 HMAC: #C1217, #C1218, #C1219, #C1222	KTS[1] (AES)	[FIPS 198-1] [FIPS180-4] [SP800-67] [SP800-38A] [SP800-38C] [SP800-38D] [SP800-38F]	AES-GCM AES-CCM AES-KW AES-CBC+HMAC-SHA1/224/256 / 384/512 Triple-DES+HMAC-SHA1/224/256 / 384/512	AES keys: 128, 192, 256 bits Triple-DES keys: 192 bits HMAC keys: 112 bits and larger	Key wrapping and unwrapping

Table 7 – Cryptographic Algorithms for Intel® Xeon® E5 Processor

[1] Approved per IG D.9

3.3.2. 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
RSA encryption / decryption	Key encapsulation with keys between 2048 and 4096 bits

Table 8 – 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
AES-XTS with 192-bit keys	“xts”	Data Encryption / Decryption
2-key Triple-DES	“des3_ede”, ”cmac(des3_ede)”	Data Encryption / Decryption
Generic GCM encryption with external IV	“gcm(aes)” with external IV	Data Encryption (certs. #C1220, #C1221, #C1222)
RFC4106 GCM encryption with external IV	“rfc4106(gcm(aes))” with external IV	Data Encryption (certs. #C1211, #C1213, #C1215, #C1303)
GHASH	“ghash”	Hashing outside the GCM mode
HMAC with less than 112 bits key	“hmac”	Message Authentication Code
RSA sign primitive operation	“rsa”	RSA signature generation
RSA verify primitive operation with keys smaller than 1024 bits and larger than 4096 bits	“rsa”	RSA signature verification
RSA encrypt and decrypt primitive operations with keys smaller than 2048 bits	“rsa”	RSA key encapsulation
SHA-1 using multi-buffer implementations	“sha1-mb”	Message Digest
ECDSA	“ecdsa”	ECDSA key generation
EC Diffie-Hellman	“ecdh”	EC Diffie-Hellman shared secret computation

Table 9 - 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) that are used by the cryptographic services implemented in the module:

Name	CSP Type	Generation	Entry and Output	Zeroization

AES key	128, 192, 256 bits AES key	None	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 key	192 bits Triple-DES key			
HMAC key	HMAC key greater than 112 bits	None	The key is passed into the module via API input parameters in plaintext.	`crypto_free_shash()` `crypto_free_ahash()`
Entropy input string	Random number	Obtained from NDRNG	None	`crypto_free_rng()`
DRBG internal state (V, C for Hash; V, C, Key for HMAC and CTR)	DRBG internal state	During DRBG initialization	None	`crypto_free_rng()`
RSA key wrapping private key	RSA private key from 2048 to 4096 bits	None	Keys are passed into the module via API input parameters in plaintext.	`crypto_free_kpp()`

Table 10 - Life cycle of Critical Security Parameters (CSPs)

The following table summarizes the asymmetric public keys that are used by the cryptographic services implemented in the module:

Name	CSP Type	Generation	Entry and Output	Zeroization
RSA key wrapping public key	RSA public key from 2048 to 4096 bits	None	Keys are passed into the module via API input parameters in plaintext.	`crypto_free_kpp()`
RSA signature verification public key	RSA public key from 1024 to 4096 bits	None	Keys are passed into the module via API input parameters in plaintext.	`crypto_free_kpp()`

Table 11 – Life cycle of asymmetric 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 and the CPU-Jitter RNG (both within the module's logical boundary). The NDRNG provides one bit of entropy per bit, therefore the DRBG is seeded with at least 256 bits of entropy 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 for symmetric or asymmetric keys. 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, GCM, CCM mode, as well as a combination of AES-CBC+HMAC. The module also supports a combination of Triple-DES-CBC+HMAC.

The module provides also provides RSA key encapsulation.

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.
- Triple-DES key wrapping provides between 112 bits of encryption strength.
- RSA key wrapping[1] provides between 112 and 149 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 and asymmetric 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.

[1] "Key wrapping" is used instead of "key encapsulation" to show how the algorithm will appear in the certificate per IG G.13.

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 performs 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 9.2.4 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 RSA using a 4096-bit modulus 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 (/boot/vmlinuz-4.4.0.1017-fips file) is ensured with the HMAC-SHA-512 value stored in the .hmac file (/boot/.vmlinuz-4.4.0.1017-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, AES-192, AES-256, without PR • KAT of CTR_DRBG with AES-128 with PR
RSA	• KAT of RSA encryption primitive with 2048-bit modulus key. • KAT of RSA decryption primitive with 2048-bit modulus key. • KAT of RSA signature verification is covered by the integrity tests which is allowed by [FIPS140-2_IG] IG 9.3

Table 12- 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. The following implementations are being self-tested during boot:

- aes-generic[1], aes-asm[2], aes-aesni[3]
- des3_ede-generic, des3_ede-asm
- sha1-generic, sha1-avx[4], sha1-avx2[5], sha1-ssse3[6]
- sha224-avx, sha224-avx2, sha224-ssse3
- sha256-generic, sha256-avx, sha256-avx2, sha256-ssse3
- sha384-generic, sha384-avx, sha384-avx2, sha384-ssse3
- sha512-generic, sha512-avx, sha512avx2, sha512-ssse3
- hmac(sha1-generic), hmac(sha1-avx2)
- hmac(sha224-avx2)
- hmac(sha256-generic), hmac(sha256-avx2)
- hmac(sha384-avx2)
- hmac(sha512-generic), hmac(sha512-avx2)
- rsa-generic
- ghash-generic, ghash-clmulni[7]
- drbg_nopr_ctr_aes128, drbg_nopr_ctr_aes192, drbg_nopr_ctr_aes256, drbg_nopr_hmac_sha256, drbg_nopr_sha256, drbg_pr_ctr_aes128, drbg_pr_hmac_sha256, drbg_pr_sha256

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:

[1] generic = C implementation

[2] asm = assembly implementation

[3] aesni = AES-NI implementation

[4] avx = Advanced Vector eXtention for Intel processor

[5] avx2 = Advanced Vector eXtension 2 for Intel processor

[6] ssse3 = Supplemental Streaming SIMD Extensions 3 (SSSE3 or SSE3S) for Intel processor

[7] clmulni = AES-NI implementation of GHASH

Algorithm	Conditional Test
NDRNG	• CRNGT

Table 13 - 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_0.0.5.2_amd64.deb linux-fips_4.4.0.1017.22_amd64.deb

Table 14 – Debian packages

9.1.1. Module Installation

The Crypto Officer can install the Debian packages containing the module listed in Table 14 using the 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 email "sales@canonical.com" or contact a Canonical representative, https://www.ubuntu.com/contact-us.

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.

- Create the file `/etc/default/grub.d/99-fips.cfg` with the content: `GRUB_CMDLINE_LINUX_DEFAULT="$GRUB_CMDLINE_LINUX_DEFAULT fips=1"`.

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

(3) Update the boot loader.

- Execute the `update-grub` 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.

Note: The module implements the IPsec protocol which has not been reviewed or tested by the CAVP and CMVP.

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.

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

9.2.3. Triple-DES

Data encryption using the same three-key Triple-DES key shall not exceed 2^{16} Triple-DES 64-bit 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
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
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**
October 23, 2019
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

