



# Microsoft FIPS 140 Validation

## Microsoft Azure Networking Adapter Kernel

*Software Version 1.0, Hardware Version BCM58732*

*Non-Proprietary*

# Security Policy Document

Document Version Number	1.0
Updated On	September 1, 2021

*The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.*

*This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.*

*Complying with all applicable copyright laws is the responsibility of the user. This work is licensed under the Creative Commons Attribution-NoDerivs-NonCommercial License (which allows redistribution of the work). To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd-nc/1.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.*

*© 2021 Microsoft Corporation. All rights reserved.*

*The names of actual companies and products mentioned herein may be the trademarks of their respective owners.*

**Version History**

Document Version	Date	Summary of changes
<b>1.0</b>	September 1, 2021	Draft sent to NIST CMVP

TABLE OF CONTENTS

**SECURITY POLICY DOCUMENT .....1**

**VERSION HISTORY.....3**

**1 INTRODUCTION.....6**

1.1 LIST OF CRYPTOGRAPHIC MODULE LIBRARIES AND BINARIES .....6

1.2 VALIDATED PLATFORMS .....7

1.3 MODES OF OPERATION.....7

1.4 CRYPTOGRAPHIC BOUNDARY.....8

1.5 FIPS 140-2 APPROVED ALGORITHMS .....9

1.6 NON-APPROVED ALGORITHMS .....10

1.7 HARDWARE COMPONENTS .....10

**2 CRYPTOGRAPHIC MODULE PORTS AND INTERFACES .....11**

**3 ROLES, SERVICES, AND AUTHENTICATION .....12**

3.1 ROLES.....12

3.2 SERVICES .....12

3.3 AUTHENTICATION .....14

**4 FINITE STATE MODEL.....15**

4.1 STATE DESCRIPTIONS.....16

**5 OPERATIONAL ENVIRONMENT.....16**

5.1 SINGLE OPERATOR .....16

5.2 TRACING .....16

**6 CRYPTOGRAPHIC KEY MANAGEMENT .....16**

6.1 RANDOM NUMBER GENERATION .....16

6.2 KEY AND CSP MANAGEMENT SUMMARY .....17

6.3 KEY AND CSP ACCESS.....18

6.4 KEY AND CSP STORAGE .....18

**6.5 KEY AND CSP ZEROIZATION .....18**

**7 SELF-TESTS .....18**

**7.1 POWER-ON SELF-TESTS .....18**

7.1.1 INTEGRITY TESTS..... 18

7.1.2 CRYPTOGRAPHIC ALGORITHM TESTS ..... 19

**7.2 ON-DEMAND SELF-TESTS.....19**

**7.3 CONDITIONAL TESTS.....19**

7.3.1 DRBG ..... 19

7.3.2 ENT ..... 20

7.3.3 AES-XTS ..... 20

**8 GUIDANCE .....20**

**8.1 CRYPTO-OFFICER GUIDANCE .....20**

8.1.1 MODULE INSTALLATION AND OPERATING ENVIRONMENT CONFIGURATION..... 20

**8.2 USER GUIDANCE .....21**

8.2.1 AES ..... 21

8.2.1.1 AES-XTS ..... 21

**8.3 HANDLING FIPS-RELATED/SELF-TEST ERRORS.....21**

**9 MITIGATION OF OTHER ATTACKS.....21**

**10 SECURITY LEVELS.....21**

**11 ADDITIONAL DETAILS .....22**

**12 GLOSSARY AND ABBREVIATIONS .....22**

**13 REFERENCES.....22**

## 1 Introduction

The Microsoft Azure Networking Adapter Kernel 1.0 (the “module”) is a general-purpose, software-hybrid cryptographic module. The module provides general purpose cryptographic services that leverage FIPS 140-2-approved cryptographic algorithms. The module runs as part of the operating system kernel, provides cryptographic services to kernel applications through a C language Application Program Interface (API), and provides cryptographic services to user applications through an AF\_ALG socket-type interface. The module is implemented as a set of shared libraries and binary files.

### 1.1 List of Cryptographic Module Libraries and Binaries

The module includes the following libraries and binaries:

Description	Library or Binary
<b>Integrity check binary file</b>	/usr/bin/sha512hmac
<b>Integrity check binary HMAC file</b>	/usr/bin/.sha512hmac.hmac
<b>Static kernel binary</b>	Part of FIT Image payload in OS-A/OS-B partitions
<b>Static kernel binary HMAC file</b>	/boot/.Image.hmac


The libraries and binaries listed are delivered within the payload of the immutable OS image on the validated platform. The Flattened ulmage Tree (FIT) image contains a kernel, an initial in-memory root file system (initramfs), and a full root file system. The initramfs extracts the kernel binary from the FIT image payload into /boot/Image.

The following components are leveraged for integrity checking, as described in more detail in the section, [Integrity Tests](#):

Component	Description
kernel-5.10	Provides the binary files and integrity check HMAC file for the kernel.
libkcapi	Provides the sha512hmac binary file that verifies the integrity of both the sha512hmac file and the image (static kernel binary) file.

## 1.2 Validated Platforms

The module has been validated on the following platform:

Processor and Platform	Processor Architecture	Operating System	Processor Image
Broadcom SoC 8 Core ARM v8 Cortex A72, with PAA (AES and SHA extensions), version BCM58732, installed in the Azure Compute C2030 Server	ARM Architecture 64-bit	Immutable OS version 1.0 (build 5.10.54.4-microsoft-standard-2008.3.21082301)	

## 1.3 Modes of Operation

The module supports three modes of operation:

1. **FIPS approved mode:** This is the approved mode of operation. In this mode, only approved security functions with sufficient security strength can be used.
2. **Non-FIPS approved mode:** This is the non-approved mode of operation. In this mode, only non-approved security functions can be used.
3. **FIPS approved mode with DRBG and CPU Jitter Entropy unavailable:** same as “approved mode” but DRBG and Jitter Entropy APIs return an error.

The module enters the FIPS approved mode after Power-On Self-Tests (POST) succeed. If the POST or Conditional Tests fail, the module goes into an error state. The status of the module can be determined by the availability of the module. If the module is available, then it has passed all self-tests; if it is not available, then it has not passed all self-tests.

If the DRBG or CPU jitter entropy self-tests encounter permanent errors, the module enters a second type of error state. The status of the module can be determined by calling the APIs for instantiating and/or generating random numbers using DRBG or CPU jitter entropy: these APIs return an error value. During this second type of error state, cryptographic services other than DRBG or CPU jitter entropy continue to function in approved mode.

A non-approved algorithm or an approved algorithm with a non-approved key size will result in the module implicitly entering the non-FIPS approved mode of operation. Critical Security Parameters (CSPs) used or stored in FIPS approved mode are not used in the non-FIPS approved mode, and vice versa.

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.

### 1.4 Cryptographic Boundary

Figure 1 shows the software block diagram for the module, its interfaces with the operational environment and logical boundary. As a software-hybrid, the module consists of disjoint software and hardware components within the same physical boundary of the host platform (see [Hardware Components](#) for additional information). The software components are the binaries listed in the section, [List of Cryptographic Module Libraries and Binaries](#), and the hardware components are the CPUs running on the host platform. All components in the orange box are included in the module.

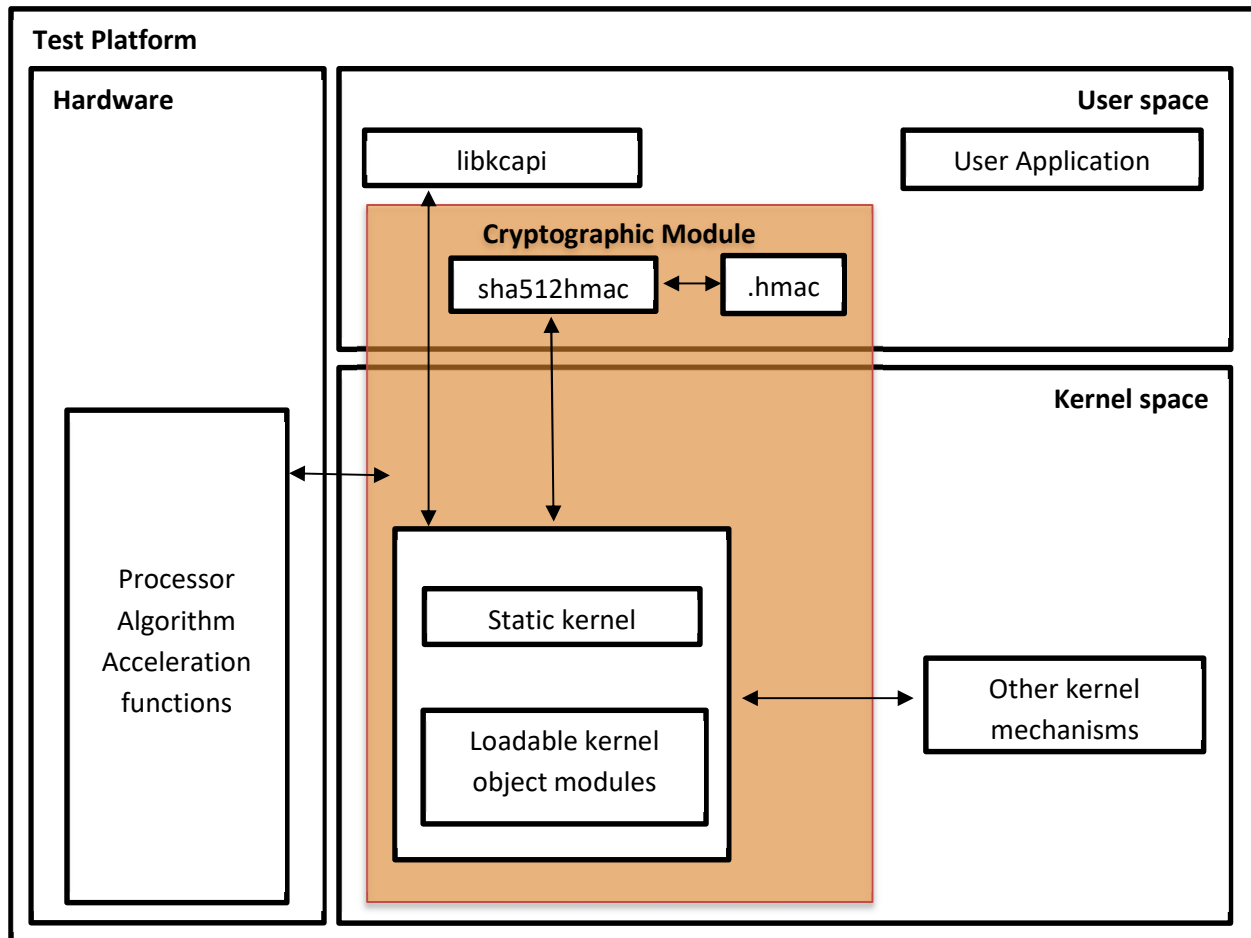


Figure 1: Software Block Diagram



Figure 2 shows the logical boundary of the module.

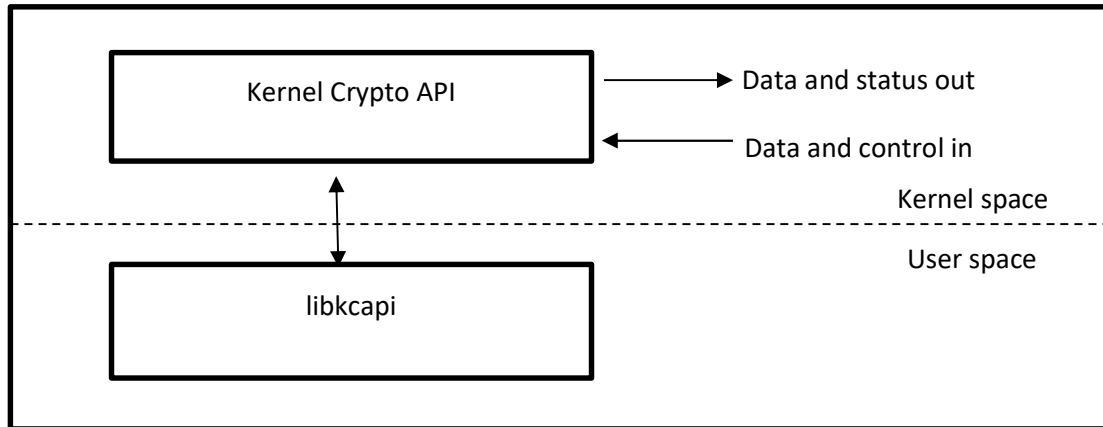


Figure 2: Kernel Cryptographic Module Logical Boundary

### 1.5 FIPS 140-2 Approved Algorithms

The following table presents the FIPS-140-2 Approved algorithms that the module may use in FIPS mode, along with the CAVP certificate that covers each:<sup>1</sup>

Algorithm	Purpose	Standards (Modes, Methods)	Key Size(s)	CAVP Certificate
<b>AES</b>	Symmetric Encryption and Decryption	FIPS 197 (AES) NIST SP 800-38A (CBC, CTR, ECB) NIST SP 800-38E (XTS)	128, 192 and 256-bit AES keys  (XTS mode only with 128 and 256-bit keys)	<a href="#">A1977</a>
<b>KTS</b>	Key Wrapping	SP 800-38A (AES-CBC, CTR, ECB) with FIPS 198-1 (HMAC-SHA-1, HMAC-SHA256, HMAC-SHA-512)	Refer to AES and HMAC key sizes	AES cert. <a href="#">A1977</a> and HMAC cert. <a href="#">A1977</a> <sup>2</sup>
<b>SHA</b>	Message Digest (SHS)	FIPS 180-4 (SHS SHA-1, SHA-256, SHA-384, SHA-512)	N/A	<a href="#">A1977</a>
<b>HMAC</b>	Keyed Hash	FIPS 198-1 (HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-512)  BS < KS, KS = BS, KS > BS	At least 112 bits HMAC keys	<a href="#">A1977</a>

<sup>1</sup> This module may not use some of the capabilities described in each CAVP certificate. Per FIPS 140-2 IG G.5, the CMVP makes no statement as to the correct operation of the Module or the security strengths of the generated keys when those are ported and executed in an operational environment not listed on the validation certificate.

<sup>2</sup> Key establishment methodology provides between 128 and 256 bits of encryption strength

<b>RSA</b>	Signature Verification	FIPS 186-4 PKCS#1 v1.5	2048, 3072, and 4096-bit signature verification, using SHA-256, SHA-384, SHA-512	<a href="#">A1977</a>
	Signature Generation	FIPS 186-4 PKCS#1 v1.5 ( SHA-256, SHA-384, SHA-512)	2048, 3072, 4096-bit RSA private key	<a href="#">A1977</a>
<b>DRBG</b>	Random Bit Generation	NIST SP 800-90A HMAC Mode DRBG (With and without prediction resistance function using SHA-1, SHA-256, SHA-384 and SHA-512)	N/A	<a href="#">A1977</a>
<b>ENT (NP)</b>	Entropy Source	SP 800-90B	N/A	N/A

## 1.6 Non-Approved Algorithms

The following table presents the non-approved algorithms implemented by the module, and the purpose for which they are used.

Algorithm	Purpose	Modes, Methods, Notes
<b>AES</b>	Symmetric encryption/ decryption	XTS with 192-bit AES keys
<b>SHA-1 (multiple-buffer implementation)</b>	Message Digest (SHS)	N/A
<b>HMAC</b>	Keyed Hash (HMAC)	HMAC keys smaller than 112 bits
<b>RSA</b>	Signature Generation	Using SHA-1, RSA private key
<b>ansi_cprng</b>	Random number generation (seed)	N/A

## 1.7 Hardware Components

The module is a multi-chip standalone software-hybrid module. As a software-hybrid, the module consists of disjoint software and hardware components within the same physical boundary of the host platform. The module runs on an ARMv8 processor as part of a System on a Chip (SOC). The host platform (SOC) meets the FIPS 140-2 level 1 Physical Security requirement. The host platform consists of production-grade components that include standard passivation techniques and is entirely contained within a metal or hard plastic production-grade enclosure that may include doors or removable covers. The physical boundary of the module is the surface of the case of the tested platform. Figure 3 shows this definition of the boundary and illustrates the hardware components used by the module.

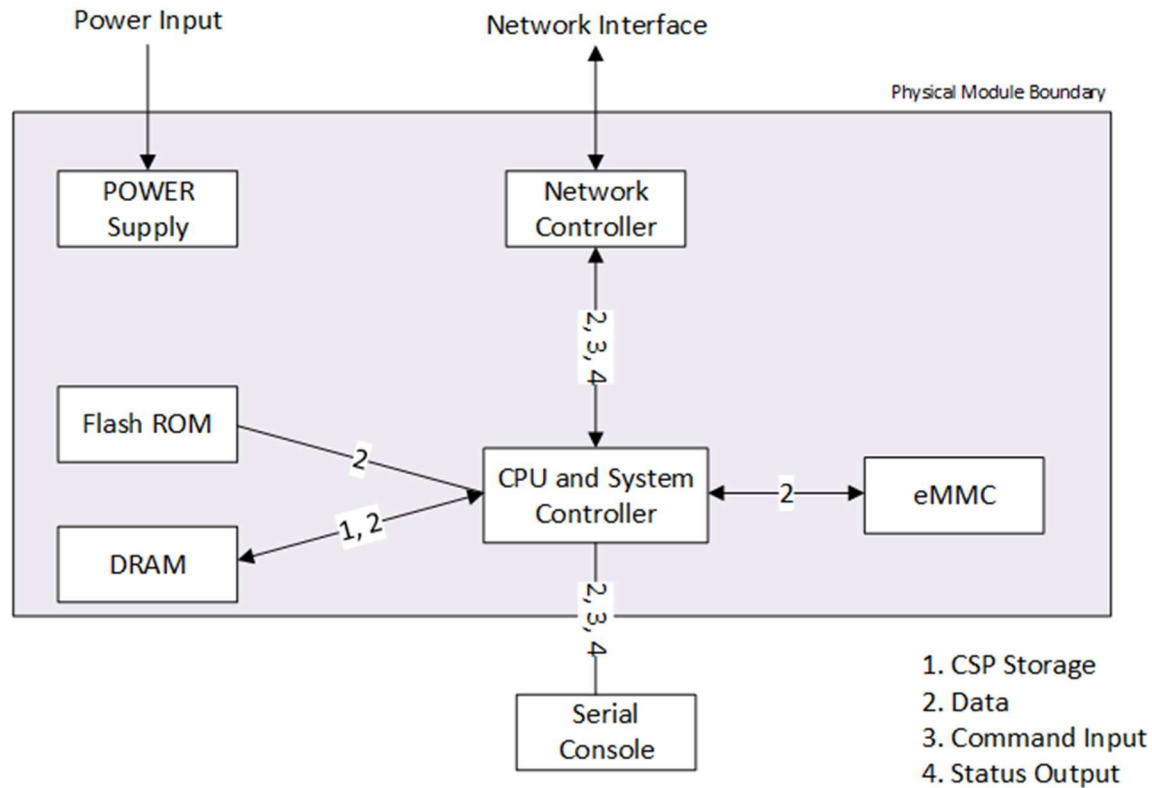


Figure 3: Kernel Cryptographic Module Physical Boundary

## 2 Cryptographic Module Ports and Interfaces

The module is a software-hybrid module. The physical ports of the module are interpreted as those on the underlying hardware platform. The logical interfaces are the application program interface (API) through which applications request services. The table below summarizes the logical interfaces and the physical ports they leverage:

Logical Interface	Physical Port	Description
Data Input	Serial Console	API input parameters from the kernel system calls AF_ALG type socket.
Data Output	Serial Console	API output parameters from the kernel system calls AF_ALG type socket.
Control Input	Serial Console	API function calls API input parameters for control from kernel system calls AF_ALG type socket kernel command line.
Status Output	Serial Console	API return codes AF_ALG type socket kernel logs.
Power Input	GPC Power Supply Port	N/A

### 3 Roles, Services, and Authentication

#### 3.1 Roles

The module supports two roles: user and crypto officer. The user and crypto officer roles are implicitly assumed by the entity accessing the module services.

- **User role:** performs all services except module installation.
- **Crypto officer role:** performs module installation and configuration.

#### 3.2 Services

The following tables provide a mapping of the available services, algorithms, Critical Security Parameters, and access types that the module provides in its different modes.

Table 5 presents the FIPS 140-2 Approved services available in FIPS mode. See the section, [FIPS 140-2 Approved Algorithms](#), for the CAVP certificate details for each algorithm.

Service	Algorithms	Notes/Modes	CSPs	Role	Access
<b>Symmetric encryption/decryption</b>	AES	CBC, CTR, ECB, XTS	128, 192 and 256 bits AES keys (XTS mode only with 128 and 256 bits keys)	User	Read
<b>Message Digest (SHS)</b>	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	N/A	User	N/A
<b>Keyed Hash (HMAC)</b>	HMAC SHA-256, HMAC SHA-512	BS < KS, KS = BS, KS > BS	At least 112 bits HMAC keys	User	Read
<b>Signature Verification</b>	RSA	2048, 3072, and 4096 bits signature verification according to PKCS#1 v1.5 using SHA-256, SHA-384, SHA-512	N/A	User	Read
<b>Signature Generation</b>	RSA	SHA-256, SHA-384, SHA-512	2048, 3072, 4096-bit RSA private key	User	Read

<b>Authenticated Encryption (KTS)</b>	AES-CBC, HMAC SHA-256, HMAC SHA-512	CBC and HMAC used with encrypt-then-MAC cipher (authenc) used for IPsec	128, 192 and 256 bits AES keys, HMAC keys	User	Read
<b>Random bit generation (SP 800-90A DRBG)</b>	HMAC DRBG	With and without prediction resistance function using SHA-1, SHA-256, SHA-384 and SHA-512	Entropy input string, seed, V, C values and Key (K)	User	Read, Write
<b>Entropy source (SP 800-90B)</b>	ENT	CPU time jitter entropy source	N/A	User	N/A
<b>Self-Tests</b>	HMAC SHA-512, RSA signature verification	Integrity test of the kernel static binary performed by HMAC SHA-512; RSA signature verification performs the integrity check of the entire block device containing the loadable kernel modules.	N/A	User	N/A
<b>Show status</b>	N/A	Via verbose mode, exit codes and kernel logs (dmesg)	N/A	User	N/A
<b>Zeroization</b>	N/A	N/A	All CSPs	User	N/A
<b>Installation and Configuration</b>	N/A	N/A	N/A	Crypto Officer	N/A

*Table 5: Services in the FIPS-Approved Mode of Operation*

Table 6 presents the non-FIPS Approved services that the module supports, which may not be used in the FIPS Approved mode of operation:

Service	Algorithms	Notes/Modes	CSPs	Access
<b>Symmetric encryption/decryption</b>	AES	XTS with 192-bit keys	192 bits AES keys	Read
<b>Message Digest (SHS)</b>	SHA-1 (multiple-buffer implementation)	N/A	N/A	Read
<b>Signature Generation</b>	RSA	Using SHA-1	RSA private key	Read
<b>Random number generation</b>	ansi_cprng	N/A	seed	Read, Write

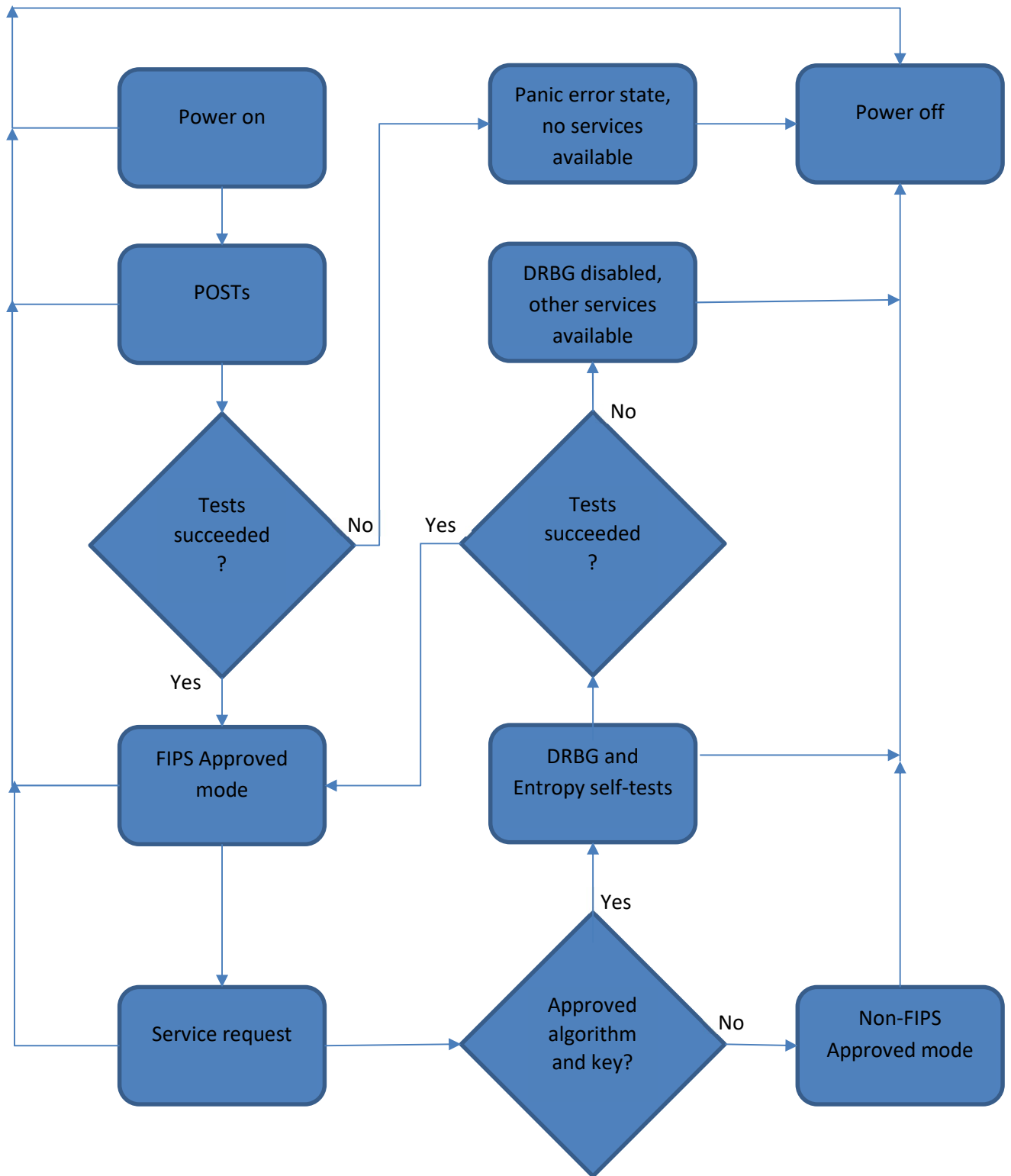
*Table 6: Services in the Non FIPS-Approved Mode of Operation*

### 3.3 Authentication

The module does not provide authentication of users. Roles are implicitly assumed based on the services that are executed.

### 4 Finite State Model

The following diagram presents the module’s operational and error states.



## 4.1 State Descriptions

The module has nine distinct states, as shown in the diagram above and described in the list below.

- 1) **Power-On State:** The module transitions to the Power-On state when the module (kernel) is loaded into memory by the bootloader.
- 2) **Power-On Self-Test (POST) State:** After being loaded, the module enters the POST state when “fips=1” is set on the Linux kernel command line and the execution of the kernel begins. The POST state will execute the integrity tests as well as the self-tests. Depending on the test results, the module will either enter FIPS Approved mode state or the Panic error state..
- 3) **Panic error state:** The POST failed or a conditional test failed. No crypto operations may be performed. The module will terminate upon further use.
- 4) **FIPS Approved mode:** The POST passed. Approved cryptographic services can now be used.
- 5) **Service request state:** a caller is requesting a cryptographic service from the module.
- 6) **Non-FIPS Approved mode:** a non-Approved algorithm or key has been used during a service request.
- 7) **DRBG and Entropy self-tests:** The module is executing DRBG and/or entropy self-tests. Depending on the test results, the module will return to FIPS Approved mode or enter the DRBG disabled state.
- 8) **DRBG disabled:** Same as FIPS approved mode but the DRBG is disabled.
- 9) **Power-off state:** the hardware and module has been shut down. The following diagram presents the module’s operational and error states.

## 5 Operational Environment

The module operates in a modifiable operational environment per FIPS 140-2 Level 1 specifications. The module runs on a general-purpose operating system executing on the hardware specified in the [Validated Platforms](#) section.

### 5.1 Single Operator

The underlying operating system of the module is restricted to a single operator. The application that requests cryptographic services is the single user of the module.

### 5.2 Tracing

In FIPS Approved mode, the ptrace system call, the debugger (gdb) and other tracing mechanisms such as ftrace or systemtap shall not be used.

## 6 Cryptographic Key Management

### 6.1 Random Number Generation

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



The module supports the HMAC\_DRBG mechanism, with security strength 128 and 256. For seeding, the module uses a number of entropy input bits equal to 1.5 times the security strength of the DRBG algorithm. For reseeding, it uses a number of entropy input bits equal to the security strength of the DRBG algorithm. The entropy input bits are obtained from a SP800-90B compliant CPU time Jitter RNG, implemented within the module's logical boundary.

The module creates a personalization string obtained from the Linux RNG. An application using the DRBG can provide a second personalization string. The bits from both of these personalization strings are used for seeding the DRBG, together with the entropy input from the CPU time Jitter RNG.

## 6.2 Key and CSP Management Summary

The following table summarizes the management of keys or other CSPs by the module.

Key or CSP	Key Generation	Key Entry and Output	Key Zeroization
<b>AES symmetric keys</b>	N/A	Keys are passed to the module via API input parameters	Memory is automatically overwritten by zeroes when freeing the cipher handler
<b>SP 800-90B DRBG seed material and internal state values V, C, and K</b>	Derived from entropy input as defined in SP800-90A	N/A	Memory is automatically overwritten by zeroes when freeing the cipher handler
<b>HMAC keys</b>	N/A	HMAC key can be supplied by calling application	Memory is automatically overwritten by zeroes when freeing the cipher handler
<b>RSA private key</b>	N/A	Keys are passed to the module via API input parameters	Memory is automatically overwritten by zeroes when freeing the cipher handler
<b>RSA public key</b>	N/A	Keys are passed to the module via API input parameters	Memory is automatically overwritten by zeroes when freeing the cipher handler
<b>Static Kernel image integrity using HMAC</b>	N/A	Key built into the sha512hmac binary during its compilation	Memory is overwritten with zero values when the sha512hmac application exits
<b>Loadable kernel component RSA public key</b>	N/A	RSA public key loaded from a keyring file in /proc/keys/	Memory is overwritten with zero values after the signature verification

### 6.3 Key and CSP Access

When an authorized application is the module user (the User role), it has access to all key data generated during the operation of the module.

### 6.4 Key and CSP Storage

Symmetric and asymmetric keys are provided to the module by the appropriate API input parameters and are destroyed when released by the appropriate API function calls.

The module does not perform persistent storage of keys. Most keys and CSPs are stored as plaintext in the RAM. RSA public key used for signature verification of the kernel loadable components is stored outside of the module's boundary, in a keyring file in `/proc/keys/`.

### 6.5 Key and CSP Zeroization

The application that uses the module is responsible for appropriate destruction and zeroization of the key material. The memory occupied by keys is allocated by regular memory allocation operating system calls. The module provides functions for key allocation and destruction, which overwrites the memory that is occupied by the key information with "zeros" before it is deallocated.

## 7 Self-Tests

FIPS 140-2 requires that the module perform self-tests to ensure the integrity of the module and the correctness of the cryptographic functionality at start up. If any self-test fails, it panics the Module, which then enters an error state. In an error state, no data output or cryptographic operations are allowed. The only recovery is to reboot. For persistent failures, you must reinstall the kernel. No user intervention is required during the running of the self-tests.

If permanent errors are encountered by the DRBG or CPU jitter entropy self-tests, the Module enters a second type of error state. During this error state, APIs related to DRBG and CPU Jitter entropy return failure error codes, but other services continue to work in approved mode.

### 7.1 Power-On Self-Tests

The module performs power-up self-tests at module initialization to ensure that the module is not corrupted and that the cryptographic algorithms work as expected. The self-tests are performed without any user intervention.

While the module is performing the power-up tests, services are not available and neither input nor output is possible. The module will not return to the calling application until the power-up self-tests are completed successfully.

#### 7.1.1 Integrity Tests

The module verifies its integrity through with an HMAC SHA-512 calculation that is performed on the `sha512hmac` utility and static kernel binary. Any additional code components loaded into the kernel are

checked with the RSA signature verification implementation of the kernel when loading them into the kernel to confirm their integrity.

The kernel integrity check passing, which requires the loading of sha512hmac with the self tests, implies a successful execution of the integrity and self tests of sha512hmac (the HMAC is stored in /usr/bin/sha512hmac.hmac).

With respect to the integrity check of kernel loadable components providing the cryptographic functionality, the fact that the self test of these cryptographic components are displayed implies that the integrity checks of each kernel component passed successfully. The integrity check for the loadable components uses a RSA2048 signature of the SHA256 root hash of a Merkle tree to verify the entire root filesystem image that contains the loadable components.

### 7.1.2 Cryptographic Algorithm Tests

The table below summarizes the power-on self tests performed by the module, which includes the Integrity Test of the module itself as stated above and the Known Answer Test for each approved cryptographic algorithm. The health tests are the RCT and APT (see [ENT](#)).

Algorithm	Power-Up Tests
<b>AES (CBC, CTR, and ECB modes: 128, 192, and 256-bit keys; XTS mode: 128 and 256-bit keys)</b>	KAT, encryption/decryption tested separately
<b>RSA signature generation (RSA-2048 with SHA-256)</b>	KAT
<b>RSA signature verification (RSA-2048 with SHA-256)</b>	KAT, also covered by loadable kernel components integrity test
<b>DRBG (HMAC using SHA-256) for instantiate, generate, and reseed</b>	KAT
<b>HMAC SHA-256, HMAC SHA-512</b>	KAT
<b>SHA-1, SHA-256, SHA-384, SHA-512</b>	KAT
<b>Static kernel image integrity check</b>	KAT, verify expected HMAC SHA-512 value
<b>Entropy source</b>	SP 800-90B, Section 4.3 startup health tests

## 7.2 On-Demand Self-Tests

The Crypto Officer with physical or logical access to the Module can run the POST (Power-On Self-Tests) on demand by power cycling the computer or by 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.

## 7.3 Conditional Tests

### 7.3.1 DRBG

The module performs DRBG health tests as defined in section 11.3 of [SP800-90A] including Instantiate, Generate, and Reseed. These tests are run for each DRBG type (HMAC).

### 7.3.2 ENT

The SP800-90B Repetition Count Test (RCT, 4.4.1) and Adaptive Proportion Test (APT, 4.4.2) are performed for the CPU time jitter entropy source.

### 7.3.3 AES-XTS

The module implements the Key\_1  $\neq$  Key\_2 test, per IG A.9.

## 8 Guidance

### 8.1 Crypto-Officer Guidance

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

#### 8.1.1 Module Installation and Operating Environment Configuration

Crypto Officers use these installation instructions to install the module in their environment. The version number of the validated OS image is listed in the section, [List of Cryptographic Module Libraries and Binaries](#). Details on how the integrity of the image is checked are provided in the sections, [List of Cryptographic Module Libraries and Binaries](#) and [Integrity Tests](#).

To configure the operating environment to support FIPS, perform the following steps.

- Install a firmware build that contains a bootloader which specifies the fips=1 kernel command line parameter when booting the kernel.
  - If such a firmware build is unavailable, the FIPS validated image can be flashed to the /dev/disk/by-partlabel/OS\_A and /dev/disk/by-partlabel/OS\_B partitions:

```
# dd if=<IMAGE> of= /dev/disk/by-partlabel/OS_A
# dd if=<IMAGE> of= /dev/disk/by-partlabel/OS_B
```
  - The boot mode can be changed to PXE boot mode:

```
# bootmode set 2
```
  - Read the current kernel command line parameters:

```
# cat /proc/cmdline
```
  - Configure a tftp server on the local network to provide the FIPS validated image with the kernel command line parameters from the previous step with “ fips=1” appended to the end

- Check that the file `/proc/sys/crypto/fips_enabled` contains 1.

## 8.2 User Guidance

To run in FIPS mode, the Module must be operated using FIPS-approved services with the corresponding FIPS-approved cryptographic algorithms.

When using the Module, the user shall use memory allocation mechanisms provided by the kernel crypto API. The user shall not use the function `copy_to_user()` on any portion of the data structures used to communicate with the API. Only the cryptographic mechanisms provided with the can be used.

### 8.2.1 AES

There are three implementations of AES: `aes-generic`, `aes-arm64`, and `aes-ce` on ARM Architecture 64-bit (aarch64) machines. The additional specific implementations of AES for the aarch64 architecture are disallowed and not available on the test platforms.

#### 8.2.1.1 AES-XTS

The AES-XTS mode was designed for the cryptographic protection of data on storage devices. It must only be used for the disk encryption functionality offered by `dm-crypt`.

## 8.3 Handling FIPS-Related/Self-Test Errors

Any Module self test failure will panic the kernel and the operating system will not load. Errors occurred during the self-tests also transitions the module into the error state. Recover by rebooting the system. If the failure continues, you must reinstall the software package following the instructions in the section, [Crypto-Officer Guidance](#).

The kernel dumps self test success and failure messages into the kernel message ring buffer. After booting, 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-ce (xts(aes)) passed` " for each algorithm and sub-algorithm type.

## 9 Mitigation of Other Attacks

The module does not implement mitigation of other attacks.

## 10 Security Levels

The security level for each FIPS 140-2 security requirement is given in the following table. The module is a software-hybrid module, the host platforms of which meet the level 1 security requirement for Physical Security. See the section, [Hardware Components](#), for more information.

Security Requirement	Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	1
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A

## 11 Additional Details

For more information about FIPS 140 validations of Microsoft products, please see:

<https://docs.microsoft.com/en-us/windows/security/threat-protection/fips-140-validation>

## 12 Glossary and Abbreviations

- **AES:** Advanced Encryption Specification
- **CAVP:** Cryptographic Algorithm Validation Program
- **CSP:** Critical Security Parameter
- **DRBG:** Deterministic Random Bit Generator
- **ECB:** Electronic Code Block
- **HMAC:** Hash Message Authentication Code
- **OS:** Operating System
- **RNG:** Random Number Generator
- **RSA:** Rivest, Shamir, Addleman
- **SHA:** Secure Hash Algorithm
- **SHS:** Secure Hash Standard

## 13 References

- **FIPS 140-2 Standard**, <https://csrc.nist.gov/projects/cryptographic-module-validationprogram/standards>
- **FIPS 186-4**, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>
- **ANSI X9.52:1998**, <http://webstore.ansi.org/FindStandards.aspx?Action=displaydept&DeptID=80&Acro=X9&DpName=X9,%20Inc>
- **NIST SP 800-38E**, <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38e.pdf>
- **NIST SP 800-38F** [SP 800-38F, Block Cipher Modes of Operation: Methods for Key Wrapping | CSRC \(nist.gov\)](#)

- **NIST SP 800-90A** [SP 800-90A Rev. 1, Random Number Generation Using Deterministic RBGs | CSRC](#)
- **NIST SP 800 132** [SP 800-132, Recommendation for Password-Based Key Derivation Part 1: Storag | CSRC \(nist.gov\)](#)
- **NIST SP 800-52** [SP 800-52 Rev. 2, Guidelines for TLS Implementations | CSRC \(nist.gov\)](#)
- **NIST SP 800-131A** [SP 800-131A Rev. 2, Transitioning the Use of Crypto Algorithms and Key Lengths | CSRC \(nist.gov\)](#)