Microsoft FIPS 140 Validation

Microsoft Azure Linux Kernel Crypto API

Software Versions: 1.0 and 2.0

Non-Proprietary

Security Policy Document

Version Number	1.2
Updated On	October 04, 2023

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.

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

Version History

Version	Date	Summary of changes
1.0	September 22, 2021	First Draft
1.1	March 23, 2023	Updated to include Azure Linux Kernel version 2
1.2	October 04, 2023	Draft sent to NIST CMVP

TABLE OF CONTENTS

1 Introduction

The Microsoft Azure Linux Kernel Crypto API (the "module") is a general-purpose, software-based 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 interface. The module is implemented as a set of shared libraries and binary files. This Security Policy covers two versions of module: version 1.0, based on Azure Linux 1.1.1k-5cm1, and version 2.0, based on Azure Linux 1.1.1k-13cm2. Except for the module package name, the security policies and related design elements of the two versions are the same.

1.1 List of Cryptographic Module Libraries and Binaries

The module includes the following libraries and binaries in 1.1.1k-5cm1 (version 1.0):

Library or Binary	Description
/usr/bin/sha512hmac	Integrity check binary file
/usr/lib/hmaccalc/sha512hmac.hmac	Integrity check binary HMAC file
/boot/vmlinuz-5.10.57.1-1.cm1	Static kernel binary
/boot/.vmlinuz-5.10.57.1-1.cm1.hmac	Static kernel binary HMAC file
/lib/modules/5.10.57.1-1.cm1/kernel/crypto/af_alg.ko.xz **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/algif_aead.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/algif_hash.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/algif_rng.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/algif_skcipher.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/ansi_cprng.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/arc4.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/authenc.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/authencesn.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/blake2b_generic.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/ccm.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/cmac.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/crypto_engine.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/crypto_user.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/des_generic.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/dh_generic.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/ecc.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/ecdh_generic.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/echainiv.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/essiv.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/gcm.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/gf128mul.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/ghash-generic.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/lrw.ko.xz**	Loadable kernel cryptography components

/lib/modules/5.10.57.1-1.cm1/kernel/crypto/lzo-rle.ko.xz **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/lzo.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/md4.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/sha3_generic.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/tcrypt.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/xor.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/crypto/xxhash_generic.ko.xz** **/lib/modules/5.10.57.1-1.cm1/kernel/arch/x86/crypto/crc32c-intel.ko.xz**	

The module includes the following libraries and binaries in 1.1.1k-13cm2 (version 2.0):

Library or Binary	Description
/usr/bin/sha512hmac	Integrity check binary file
/usr/lib/hmaccalc/sha512hmac.hmac	Integrity check binary HMAC file
/boot/vmlinuz- 5.15.94.1-1.cm2	Static kernel binary
/boot/.vmlinuz 5.15.94.1-1.cm2.hmac	Static kernel binary HMAC file
/lib/modules/5.15.94.1-1.cm2/kernel/crypto/af_alg.ko.xz **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/algif_aead.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/algif_hash.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/algif_rng.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/algif_skcipher.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/ansi_cprng.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/arc4.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/authenc.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/authencesn.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/blake2b_generic.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/ccm.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/cmac.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/crypto_engine.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/crypto_user.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/des_generic.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/dh_generic.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/ecc.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/echainiv.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/essiv.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/gcm.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/gf128mul.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/ghash-generic.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/lrw.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/lzo-rle.ko.xz** **/lib/modules/5.15.94.1-1.cm2.cm1/kernel/crypto/lzo.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/md4.ko.xz**	Loadable kernel cryptography components

/lib/modules/5.15.94.1-1.cm2/kernel/crypto/sha3_generic.ko.xz **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/xor.ko.xz** **/lib/modules/5.15.94.1-1.cm2/kernel/crypto/xxhash_generic.ko.xz**	

The following packages are required for the module to operate in 1.1.1k-5cm1 (version 1.0):

Package Name	Description
kernel-5.10.57.1-1.cm1.x86_64.rpm	Provides the binary files and integrity check HMAC file for the kernel
libkcapi-1.2.0-5.cm1.x86_64.rpm	Provides the sha512hmac binary file that verifies the integrity of both the sha512hmac file and the vmlinuz (static kernel binary) file

The following packages are required for the module to operate in 1.1.1k-13cm2 (version 2.0):

Package Name	Description
kernel-5.15.94.1-1.cm2.x86_64	Provides the binary files and integrity check HMAC file for the kernel
libkcapi-1.3.1-2.cm2.x86_64	Provides the sha512hmac binary file that verifies the integrity of both the sha512hmac file and the vmlinuz (static kernel binary) file

1.2 Validated Platforms

The module has been validated on the following platforms:

Platform	Processor	Operating System	Configuration
Azure Compute C2030 Server	Intel® Xeon® Platinum 8272CL (Intel x86 64-bit)	Azure Linux 1.0	With and without AES-NI (PAA)
Virtual Machine on Azure Host Hypervisor, running on Azure Compute C2030 Server	Intel® Xeon® Platinum 8272CL (Intel x86 64-bit)	Azure Linux 1.0	With and without AES-NI (PAA)
Azure Compute C2030 Server	Intel® Xeon® Platinum 8272CL (Intel x86 64-bit)	Azure Linux 2.0	With and without AES-NI (PAA)
Virtual Machine on Azure Host Hypervisor, running on Azure Compute C2030 Server	Intel® Xeon® Platinum 8272CL (Intel x86 64-bit)	Azure Linux 2.0	With and without AES-NI (PAA)

1.3 Modes of Operation

The Microsoft Azure Linux Kernel Crypto API supports three modes of operation:

1. **FIPS approved mode (“approved mode”):** In this mode, only FIPS-approved security functions with sufficient security strength can be used.
2. **Non-FIPS approved mode (“non-approved mode”):** In this mode, non-approved security functions can also be used.
3. **FIPS approved mode with DRBG and CPU Jitter Entropy unavailable**: same as “approved mode” but DRBG and Jitter Entropy APIs return 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, including its interfaces, operational environment, and logical boundary. All components in the orange box are included in the module.

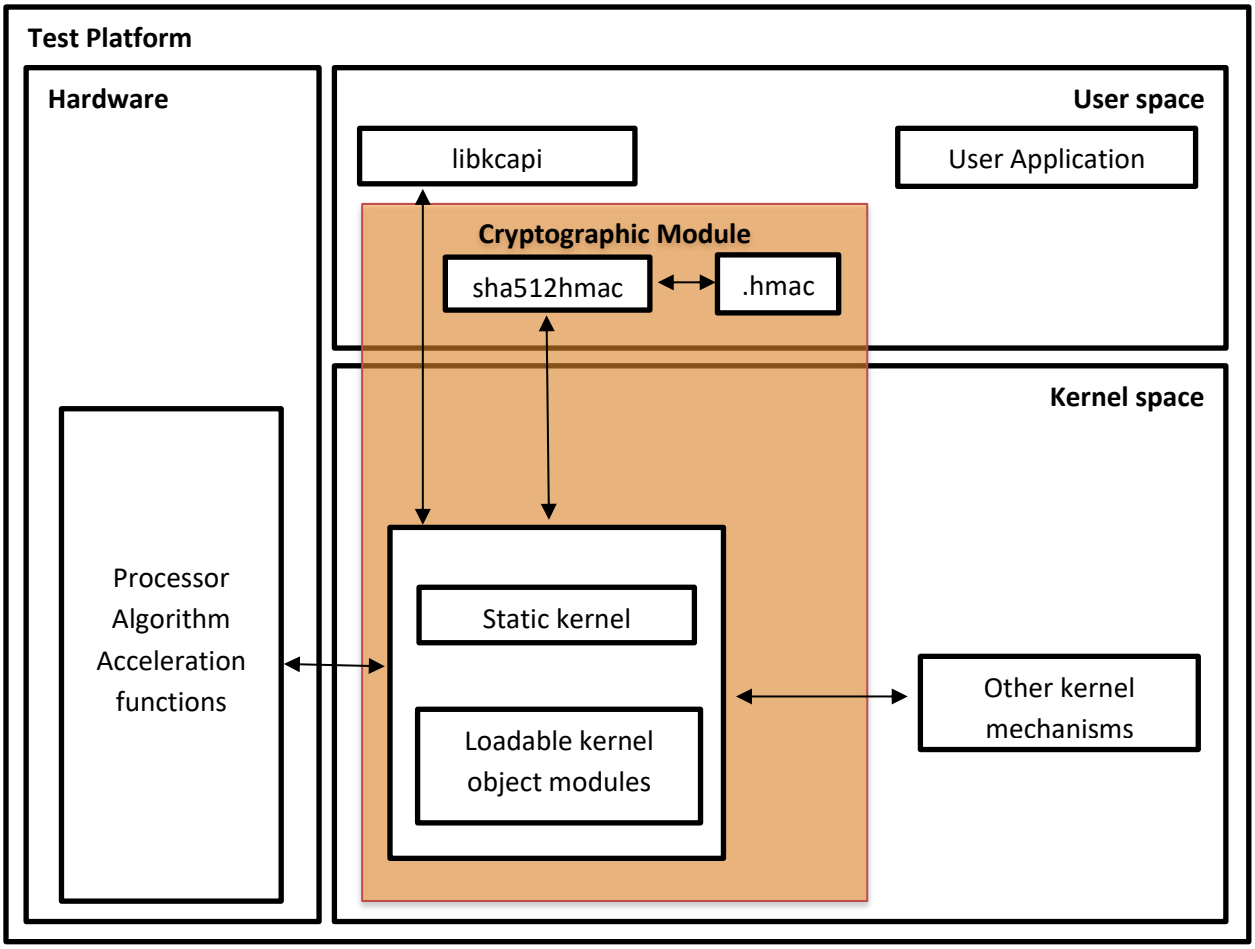

Figure 1: Module Software Block Diagram

Figure 2 shows the logical boundary of the module.

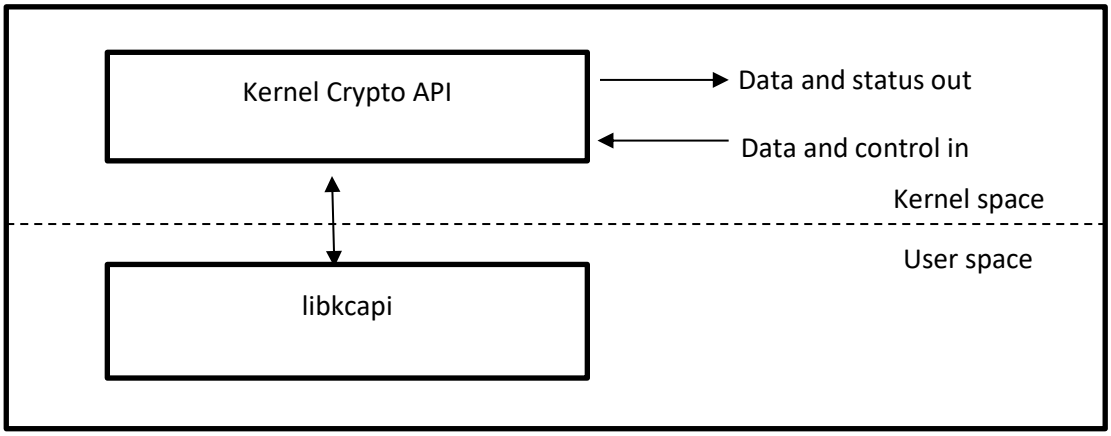

Figure 2: Module Logical Boundary

1.5 FIPS 140-2 Approved Algorithms

The following table presents the approved algorithms that the module may use in FIPS mode, along with the CAVP certificate that covers each.

Publication and Algorithm(s)	CAVP Certificate (Azure Linux Kernel) 1.1.1k-5cm1 (version 1.0)	CAVP Certificate (Azure Linux Kernel on Azure Host Hypervisor) 1.1.1k-5cm1 (version 1.0)	CAVP Certificate (Azure Linux Kernel) 1.1.1k-13cm2 (version 2.0)	CAVP Certificate (Azure Linux Kernel on Azure Host Hypervisor 1.1.1k-13cm2 (version 2.0)
FIPS 197 AES-128, AES-192, and AES-256 in ECB, CBC, and CTR modes	#A1755	#A1755	#A3494	#A3494
NIST SP 800-38B and SP 800-38C AES-128, AES-192, and AES-256 in CCM mode	#A1755	#A1755	#A3494	#A3494
NIST SP 800-38B and SP 800-38C AES-128 in CMAC mode	#A1755	#A1755	#A3494	#A3494
NIST SP 800-38D AES-128, AES-192, and AES-256 GCM	#A1755	#A1755	#A3494	#A3494

NIST SP 800-38E AES-128, AES-256 XTS Mode[1]	#A1755	#A1755	#A3494	#A3494
NIST SP 800-38F KTS	AES Cert. #A1755 and HMAC Cert. #A1755[2]	AES Cert. #A1755 and HMAC Cert. #A1755[2]	AES Cert. #A3494 and HMAC Cert. #A3494[2]	AES Cert. #A3494 and HMAC Cert. #A3494[2]
NIST SP 800-38F KTS	AES Cert. #A1755[2]	AES Cert. #A1755[2]	AES Cert. #A3494[2]	AES Cert. #A3494[2]
NIST SP 800-38F KTS	Triple-DES Cert. #A1755 and HMAC Cert. #A1755[3]	Triple-DES Cert. #A1755 and HMAC Cert. #A1755[3]	Triple-DES Cert. #A3494 and HMAC Cert. #A3494[3]	Triple-DES Cert. #A3494 and HMAC Cert. #A3494[3]
NIST SP 800-67 rev 1 Triple-DES, 168-bit in ECB, CBC, CMAC, and CFB64 Modes	#A1755	#A1755	#A3494	#A3494
FIPS 180-4 SHS SHA-1, SHA2-256, SHA2-384, and SHA2-512	#A1755	#A1755	#A3494	#A3494
FIPS 202 SHS SHA3-224, SHA3-256, SHA3-384, SHA3-512	#A1755	#A1755	#A3494	#A3494
FIPS 198-1 HMAC-SHA-1, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, and HMAC-SHA3-512	#A1755	#A1755	#A3494	#A3494
FIPS 186-4 RSA PKCS#1 v1.5 Digital Signature Verification with 2048, 3072, and 4096 Moduli; supporting SHA2-256, SHA2-384, and SHA2-512	#A1755	#A1755	#A3494	#A3494
FIPS 186-4 RSA PKCS#1 v1.5 Digital Signature Generation with 2048, 3072, and 4096 Moduli; supporting SHA2-256, SHA2-384, and SHA2-512	#A1755	#A1755	#A3494	#A3494

[1] AES XTS must be used only to protect data at rest and the caller needs to ensure that the length of data encrypted does not exceed 2^{20} AES blocks.

[2] Key establishment methodology provides between 128 and 256 bits of encryption strength

[3] Key establishment methodology provides 112 bits of encryption strength

NIST SP 800-56A rev 3 KAS-ECC (ECDH) Component (P-256), for Partial Public Key Validation (CVL)	#A1755	#A1755	#A3494	#A3494
NIST SP 800-90A AES-256 CTR Mode DRBG	#A1755	#A1755	#A3494	#A3494
NIST SP 800-90A SHA1, SHA2-256 Hash Mode DRBG	#A1755	#A1755	#A3494	#A3494
NIST SP 800-90A SHA1, SHA2-256, SHA2-384, SHA2-512 HMAC Mode DRBG	#A1755	#A1755	#A3494	#A3494
NIST SP 800-131A rev 1 RSA Signature Primitive (CVL)	#A1755	#A1755	#A3494	#A3494
NIST SP 800-90B ENT (NP)	N/A	N/A	N/A	N/A
NIST SP 800-56A rev 3 KAS-FFC-SSC, for KAS Initiation (Externally generated) p=2048-bits/q=256-bits key pairs, dhStatic scheme	#A1755	#A1755	#A3494	#A3494

1.6 Non-Approved Algorithms

The following table presents the non-approved algorithms implemented by version 2.0 based on Azure Linux 1.1.1k-13cm2, and the purpose for which they are used.

Purpose	Algorithm(s)	Notes/Modes	CSPs
Symmetric Encryption / Decryption	AES	CMAC CFB	192 and 256-bit AES keys
		GCM encryption with external IV	128, 192, 256-bit AES keys
		GMAC	128, 192, 256-bit AES keys
	DES	ECB	56 bits DES keys
Message Digest (SHS)	SHA-1 (multiple-buffer implementation)	N/A	N/A
Keyed Hash (HMAC)	HMAC	Keys smaller than 112 bits	HMAC keys with size less than 112 bits

Signature Generation	RSA	Using SHA-1	RSA private key
Key Generation	ECDSA	P-192, P-256, P384, curve25519 The PCT is not implemented.	ECDSA private key
Shared Secret Computation	KAS-ECC	Shared secret computation	KAS-ECC private keys (P-192)
	KAS-FFC		KAS-FFC private keys (smaller than 2048 bits and keys larger than 2048 bits)
Random Number Generation	ansi_cprng	N/A	seed

The following table presents the non-approved algorithms implemented by version 1.0, based on Azure Linux 1.1.1k-5cm1, and the purpose for which they are used.

Purpose	Algorithm(s)	Notes/Modes	CSPs
Symmetric Encryption / Decryption	AES	CMAC	192 and 256-bit AES keys
		XTS with 192-bit keys	192-bit AES keys
		GCM encryption with external IV	128, 192, 256-bit AES keys
		GMAC	128, 192, 256-bit AES keys
	DES	ECB	56 bits DES keys
Message Digest (SHS)	SHA-1 (multiple-buffer implementation)	N/A	N/A
Keyed Hash (HMAC)	HMAC	Keys smaller than 112 bits	HMAC keys with size less than 112 bits
Signature Generation	RSA	Using SHA-1	RSA private key
Key Generation	ECDSA	P-192, P-256	ECDSA private key

		The PCT is not implemented.	
Shared Secret Computation	KAS-ECC	Shared secret computation	KAS-ECC private keys (P-192)
	KAS-FFC		KAS-FFC private keys (smaller than 2048 bits and keys larger than 2048 bits)
Random Number Generation	ansi_cprng	N/A	seed

1.7 Hardware Components of the Cryptographic Module

The Microsoft Azure Linux Kernel Crypto API is a multi-chip standalone module. The physical boundary of the module is the physical boundary of the computer that contains the module. The following diagram illustrates the hardware components used by the module:

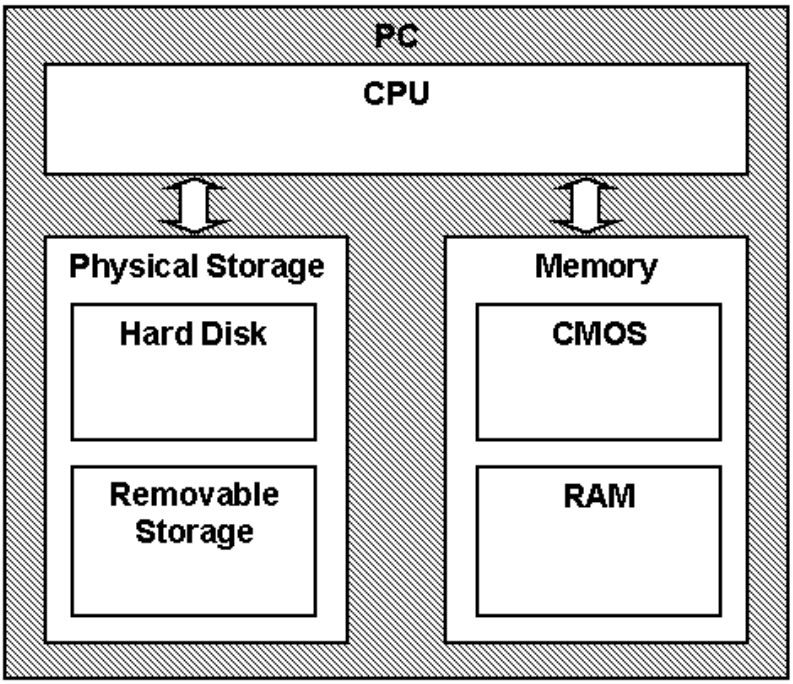

Figure 3: Module Physical Boundary / Hardware Components

2 Cryptographic Module Ports and Interfaces

The module is a software module and has no physical ports of its own. 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 describes the logical interfaces and the physical ports they leverage:

Logical Interface	Physical Port	Description
Data Input	Keyboard	API input parameters from the kernel system calls AF_ALG type socket.
Data Output	Display	API output parameters from the 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

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 FIPS 140-2 Approved Services

Table 5 provides a mapping of the available services, algorithms, Critical Security Parameters, and access types when the module is operating in FIPS Approved mode of operation. See the section, FIPS 140-2 Approved Algorithms, for the CAVP certificate details for each algorithm.

The module exposes FIPS approved services using the APIs described by https://www.kernel.org/doc/html/latest/crypto/index.html.

Service	Algorithms	Notes/Modes	CSPs	Access
Symmetric Encryption / Decryption	AES	CBC, CCM, CMAC, CTR, ECB, GCM (external IV,	128, 192 and 256 bits AES keys	Read

		decryption only), GMAC, XTS	(XTS mode only with 128 and 256 bits keys)	
		ECB, GCM (internal IV, encryption/decryption)		
		ECB, GCM (external IV, decryption only)		
	Triple-DES	CMAC, ECB, CBC, CFB 64	168 bits Triple-DES keys	
Message Digest (SHS)	SHA-1, SHA-256, SHA-384, SHA-512	N/A	N/A	N/A
	SHA3-224, SHA3-256, SHA3-384, SHA3-512			
Keyed Hash (HMAC)	HMAC SHA-1, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512	BS < KS, KS = BS, KS > BS	At least 112 bits HMAC keys	Read
	HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384 HMAC-SHA3-512			
Signature Verification	RSA	2048- and 3072-bit signature verification according to PKCS#1 v1.5 using SHA-256, SHA-384, SHA-512	N/A	Read
Signature Generation	RSA	SHA-256, SHA-384, SHA-512	2048, 3072, 4096-bit RSA private key	Read
Component Public Key Validation and Shared Secret Computation	KAS-ECC	P-256 with SHA-256, SHA-384, SHA-512	P-256-based KAS-ECC private key, Shared Secret	Read, Write
Authenticated Encryption (KTS)	AES-CBC, HMAC SHA-1, HMAC SHA-224, HMAC SHA-256, HMAC SHA-384, 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 168 bits Triple-DES keys	Read
	Triple-DES-CBC and HMAC SHA-1, HMAC SHA-224, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512			

	AES	GCM and CCM		
Random Bit Generation (SP 800-90A DRBG)	CTR DRBG	With derivation function, with and without prediction resistance function using AES-256	Entropy input string, seed, V, C values and Key (K)	Read, Write
	Hash DRBG	With derivation function, with and without prediction resistance function using SHA-1 and SHA-256		
	HMAC DRBG	With and without prediction resistance function using SHA-1, SHA-256, SHA-384 and SHA-512		
Entropy Source (SP 800-90B)	ENT	CPU time jitter entropy source	N/A	N/A
Self-Tests	HMAC SHA-512, RSA signature verification	Integrity test of the kernel static binary performed by the sha512hmac binary RSA signature verification performs the signature verification of the kernel loadable components	N/A	N/A
Show Status	N/A	Via verbose mode, exit codes and kernel logs (dmesg)	N/A	N/A
Zeroization	N/A	N/A	All CSPs	N/A
Installation and Configuration	N/A	N/A	N/A	N/A

Table 5: Services in the 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.

5 Operational Environment

The operational environment for the module is the Azure Linux operating system, running on one of the supported hardware platforms 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 Hash_DRBG, HMAC_DRBG and CTR_DRBG mechanisms, with security strengths of 128, 192 and 256. For seeding, the module uses a number of entropy input bits equal to 1.5 times the security strength of of the DRBG algorithm. For reseeding, it uses a number of entropy input bits equal to the security strength of 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 / CSP	Generation	Entry / Output	Zeroization
AES symmetric keys	N/A	Keys are passed to the module via API input parameters	Memory is automatically overwritten by zeroes when freeing the cipher handler
Triple-DES symmetric keys	N/A	Keys are passed to the module via API input parameters	Memory is automatically overwritten by zeroes when freeing the cipher handler
SP 800-90B DRBG seed material and internal state values V, C, and K	Derived from entropy input as defined in SP800-90A	N/A	Memory is automatically overwritten by zeroes when freeing the cipher handler
HMAC keys	N/A	HMAC key can be supplied by calling application	Memory is automatically overwritten by zeroes when freeing the cipher handler
KAS-FFC domain parameters	N/A	Domain parameters passed to the module via API input parameters	Memory is automatically overwritten by zeroes when freeing the cipher handler
KAS-ECC key pair	N/A	Key passed to the module via API input parameters	Memory is automatically overwritten by zeroes when freeing the cipher handler
Shared secret	Generated during the KAS-FFC or KAS-ECC shared secret computation.	Keys are passed to the module via API input parameters. Shared Secret is output in plaintext via API output.	Memory is automatically overwritten by zeroes when freeing the cipher handler
RSA private key	N/A	Keys are passed to the module	Memory is automatically overwritten by zeroes when freeing the cipher handler

		via API input parameters	
RSA public key	N/A	Keys are passed to the module via API input parameters	Memory is automatically overwritten by zeroes when freeing the cipher handler
Static Kernel image integrity HMAC	N/A	Key built into the sha512hmac binary during its compilation.	Memory is overwritten with zero values when the sha512hmac application exits.
Loaded modules signature verification public key	N/A	RSA public key loaded from a keyring file in /proc/keys/	Memory is overwritten with zero values after the signature verification.
Entropy Input	Generated internally from SP 800-90B ENT (NP)	N/A	N/A

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.The module does not support the output of intermediate key generation values during the key generation process.

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. The 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/. The KAS-FFC and KAS-ECC public keys are stored in protected kernel memory.

The kernel computes HMAC-SHA-512 values on behalf of the sha512hmac application, for the application binary file and the kernel static image image. sha512hmac compares these values with the expected HMAC values from /usr/lib/hmaccalc/sha512hmac.hmac and /boot/.vmlinuz-5.10.57.1-1.cm1.hmac in 1.1.1k-5cm1 (version 1.0) and /boot/vmlinuz- 5.15.94.1-1.cm2.hmac in 1.1.1k-13cm2 (version 2.0).

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 library provides functions for key allocation and destruction, which overwrites the memory that is occupied by the key information with zeros before it is deallocated.

6.6 Key Establishment and Transport

The module supports KAS-FFC and KAS-ECC shared secret primitive computation:

- KAS-FFC: shared secret computation provides 112 bits of encryption strength.
- KAS-ECC: shared secret computation provides 128 bits of encryption strength.

The module provides SP 800-38F compliant key wrapping using AES with GCM and CCM block chaining modes, as well as a combination of AES-CBC for encryption/decryption and HMAC for authentication. The module also provides SP 800-38F compliant key wrapping using a combination of Triple-DES-CBC for encryption/decryption and HMAC for authentication.

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 112 bits of encryption strength.

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 this 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 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 an HMAC SHA-512 calculation that is performed on the sha512hmac utility and static kernel binary. 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 expected HMAC values are stored in /usr/lib/hmaccalc/sha512hmac.hmac and

/boot/.vmlinuz-5.10.57.1-1.cm1.hmac in 1.1.1k-5cm1 (version 1.0) and /boot/vmlinuz- 5.15.94.1-1.cm2.hmac in 1.1.1k-13cm2 (version 2.0).

The static kernel loads from the keyring file in /proc/keys/ a Microsoft RSA public key corresponding to the private key used for signing kernel loadable modules. It uses that public key and kernel's RSA signature verification implementation to verify the integrity of any kernel module files that might be loaded, before allowing the execution of these modules. The fact that the self tests of these cryptographic components are displayed implies that the integrity checks of each kernel component passed successfully.

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. See the section, FIPS 140-2 Approved Algorithms, for full details on all approved cryptographic algorithms.

Algorithm	Power-On Tests
AES (ECB, CBC, CTR, CCM, CMAC, GCM, GMAC, and XTS modes)	KAT, encryption/decryption tested separately
Triple-DES (ECB, CBC, and CMAC)	KAT, encryption/decryption tested separately
RSA signature generation	KAT
RSA signature verification	KAT, also covered by integrity test
DRBG (AES CTR, Hash, HMAC)	KAT
HMAC SHA-1, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512	KAT
HMAC SHA3-224, HMAC SHA3-256, HMAC SHA3-384, HMAC SHA3-512	KAT
SHA-1, SHA-256, SHA-384, SHA-512	KAT
SHA3-224, SHA3-256, SHA3-384, SHA3-512	KAT
KAS-FFC Z primitive with 2048 bits	KAT
KAS-ECC Z primitive with P-256	KAT
Static kernel image integrity check	Verify expected HMAC SHA-512 value
Loaded modules integrity check	Verify SHA-512 signature using RSA public key

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 neither data output nor 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, CTR, Hash)

7.3.2 ENT

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

7.3.3 KAS-FFC Key Agreement Method

7.3.3.1 Owner Assurance of Public-Key Validity

This is implemented as specified by SP800-56Arev3 5.6.2.1.3.

7.3.3.2 Public Key Validation

If Q is provided as part of the domain parameters, a full validation according to SP800-56A section 5.6.2.3.1 is performed. If Q is not provided, a partial validation according to SP800-56A section 5.6.2.3.2 is performed.

7.3.3.3 DLC Primitives

The module validates the shared secret as specified by SP800-56Arev3 5.7.1.1.

7.3.4 KAS-ECC Key Agreement Protocol

7.3.4.1 Public Key Validation

The module performs partial verification for ephemeral keys, per SP800-56A section 5.6.2.3.4, and full validation for other keys, per SP800-56Arev3 5.6.2.1.3.

7.3.5 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

Crypto Officers use the Installation instructions to install the Module in their environment.

The version of the RPM containing the FIPS validated module is stated in section 1. The integrity of the RPM is automatically verified during the installation and the Crypto Officer shall not install the RPM file if the RPM tool indicates an integrity error.

8.1.2 Operating Environment Configuration

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

- Install the dracut-fips package:

```
tdnf install dracut-fips
```

- Regenerate the initramfs

  ```
  mkinitrd
  ```

- Modify the mariner.cfg file:

  Append `fips=1` to variable `mariner_cmdline` in /boot/mariner.cfg.

- Reboot the system.
- 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 API can be used.

8.2.1 CTR and RFC3686

CTR and RFC3686 mode must only be used for IPsec. It must not be used otherwise.

8.2.2 AES

There are three implementations of AES: aes-generic, aesni-intel, and aes-x86_64 on x86_64 machines. The additional specific implementations of AES for the x86 architecture are disallowed and not available on the test platforms.

8.2.2.1 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.

8.2.2.2 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.

The module generates the 96-bit IV internally randomly with an approved SP 800-90A DRBG, which is compliant with provision 2) of IG A.5.

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.

8.2.3 Triple-DES

Data encryption with the same Triple-DES key shall not exceed 2^16 64-bit blocks of data. It is the user's responsibility to make sure that the module complies with this requirement and that the module does not exceed this limit.

8.3 Handling FIPS-Related Self-Test Errors

When encountering any Power-On Self-Tests (POST) failure, the Module will panic the kernel and the operating system will not load. Errors occurred during the POST also transition the module into the error state.

If the DRBG or CPU jitter entropy self-tests encounter permanent errors, the module enters a second type of error state. During this second type of error state, cryptographic services other than DRBG or CPU jitter entropy continue to function in approved mode, but APIs related to DRBG or CPU jitter entropy return a failure error code.

Recover from any of these error states by rebooting the system. If the failure continues, you must reinstall the software package following the directions in section 10.1.

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) (xts(aes-x86_64)) 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.

Security Requirement	Security Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
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 Standard
- **CAVP:** Cryptographic Algorithm Validation Program
- **CSP:** Critical Security Parameter
- **DES:** Data Encryption Standard
- **DRBG:** Deterministic Random Bit Generator
- **DSA:** Digital Signature Algorithm
- **ECB:** Electronic Codebook
- **HMAC:** Hash Message Authentication Code
- **OS:** Operating System
- **RNG:** Random Number Generator
- **RSA:** Rivest, Shamir, Adleman
- **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)