Mirantis, Inc.

Mirantis Crypto Library

Software Version: 1.0

FIPS 140-2 Non-Proprietary Security Policy

FIPS Security Level: 1
Document Version: 1.6

Prepared for:

Mirantis, Inc.
900 E. Hamilton Avenue
Campbell, CA 95008
United States of America

Phone: +1 650 963 9828
www.mirantis.com

Prepared by:

Corsec Security, Inc.
13921 Park Center Road, Suite 460
Herndon, VA 20171
United States of America

Phone: +1 703 267 6050
www.corsec.com

Table of Contents

1. Introduction 4
1.1 Purpose 4
1.2 References 4
1.3 Document Organization 4
2. Mirantis Crypto Library 5
2.1 Product Overview 5
2.2 Module Specification 7
2.2.1 Physical Cryptographic Boundary 9
2.2.2 Logical Cryptographic Boundary 10
2.3 Module Interfaces 11
2.4 Roles and Services 12
2.5 Physical Security 14
2.6 Operational Environment 14
2.7 Cryptographic Key Management 14
2.8 EMI / EMC 17
2.9 Self-Tests 17
2.9.1 Power-Up Self-Tests 17
2.9.2 Conditional Self-Tests 17
2.9.3 DRBG Health Checks 18
2.9.4 Self-Test Failure Handling 18
2.10 Mitigation of Other Attacks 18
3. Secure Operation 19
3.1 Secure Management 19
3.1.1 Application Setup 19
3.1.2 Configuration 19
3.2 Operator Guidance 19
3.2.1 Crypto Officer Guidance 20
3.2.2 User Guidance 20
3.2.3 General Operator Guidance 20
3.3 Additional Guidance and Usage Policies 20
3.4 Non-FIPS-Approved Mode 21
4. Acronyms 22

List of Tables

Table 1 – Security Level per FIPS 140-2 Section 6
Table 2 – FIPS-Approved Cryptographic Algorithms 8
Table 3 – FIPS 140-2 Logical Interface Mappings 12

Table 4 – Operator Services 12
Table 5 – Cryptographic Keys, Cryptographic Key Components, and CSPs 15
Table 6 – Acronyms 22

List of Figures

Figure 1 – Mirantis Cloud Native Platform Components 6
Figure 2 – Host Server Physical Block Diagram 10
Figure 3 – Logical Block Diagram 11

1. Introduction

1.1 Purpose

This is a non-proprietary Cryptographic Module Security Policy for the Mirantis Crypto Library (software version: 1.0) from Mirantis, Inc. (Mirantis). This Security Policy describes how the Mirantis Crypto Library meets the security requirements of Federal Information Processing Standards (FIPS) Publication 140-2, which details the U.S. and Canadian government requirements for cryptographic modules. More information about the FIPS 140-2 standard and validation program is available on the National Institute of Standards and Technology (NIST) and the Communications Security Establishment (CSE) Cryptographic Module Validation Program (CMVP) website at http://csrc.nist.gov/groups/STM/cmvp.

This document also describes how to run the module in a secure FIPS-Approved mode of operation. This policy was prepared as part of the Level 1 FIPS 140-2 validation of the module. The Mirantis Crypto Library is also referred to in this document as the Mirantis Crypto Library and the module.

1.2 References

This document deals only with operations and capabilities of the module in the technical terms of a FIPS 140-2 cryptographic module security policy. More information is available on the module from the following sources:

- The Mirantis website (www.mirantis.com) contains information on the full line of products from Mirantis.
- The CMVP website (http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm) contains contact information for individuals responsible for answer technical or sales-related questions for the module.

1.3 Document Organization

The Security Policy document is organized into two (2) primary sections. Section 2 provides an overview of the validated modules. This includes a general description of the capabilities and the use of cryptography, as well as a presentation of the validation level achieved in each applicable functional area of the FIPS standard. It also provides high-level descriptions of how the modules meet FIPS requirements in each functional area. Section 3 documents the guidance needed for the secure use of the module, including initial setup instructions and management methods and policies.

2. Mirantis Crypto Library

2.1 Product Overview

The Mirantis Cloud Native Platform provides businesses with the ability to deploy a Containers[1] as a Service (CaaS) environment to build, ship and run applications (apps). It enables IT operations to secure, provision, and manage both infrastructure resources and base app content while allowing developers to build and deploy their apps in a self-service manner.

The Mirantis Cloud Native Platform primarily comprises three components (see Figure 1 debajo de):

- Mirantis Container Runtime – The Mirantis Container Runtime is a lightweight container runtime that allows packaging of application code and dependencies together in an isolated container that share the operating system (OS) kernel on the host system. Designed for enterprise development and IT[2] teams who build, ship, and run business critical applications in production at scale, Mirantis Container Runtime runs on both Linux and Windows OS on any infrastructure whether it's physical, virtual, or cloud.

  The Mirantis Container Runtime is comprised of a CLI[3], REST[4] API[5], and Mirantis daemon process. The CLI uses the REST API to control or interact with the Mirantis daemon process through scripting or direct CLI commands.

- Mirantis Kubernetes Engine – Mirantis Kubernetes Engine is the enterprise-grade cluster management solution from Mirantis. It enables teams to manage and deploy applications, which can run on any private infrastructure or public cloud. Some of its key features include:
  - GUI[6] management for apps, containers, nodes, networks, images and volumes
  - Monitoring and logging of Mirantis Kubernetes Engine users and events
  - LDAP[7]/AD[8] integration
  - Role-based access control (RBAC)
  - Out-of-the-box high availability
  - Push/pull images from DTR
  - Out-of-the-box TLS[9]

- Mirantis Secure Registry – Mirantis Secure Registry is the enterprise-grade image storage solution from Mirantis. It allows enterprise IT users to store and secure their images on-premises or within their virtual private cloud. Some of its key features include:
  - GUI for administrators and users
  - Monitoring and logging of DTR users

[1] Containers package an application in a complete filesystem that contains everything it needs to run - code, runtime, system tools, and system libraries. This allows the app to run the same, regardless of the environment it is running in.
[2] IT – Information Technology
[3] CLI – Command Line Interface
[4] REST – Representational State Transfer
[5] API – Application Programming Interface
[6] GUI – Graphical User Interface
[7] LDAP – Lightweight Directory Access Protocol
[8] AD – Active Directory
[9] TLS – Transport Layer Security

- LDAP/AD integration
- RBAC
- Out-of-the-box high availability for DTR instances via replicas
- Mirantis Content Trust image signing
- Garbage collection for removal of orphaned images
- Installs directly into the Mirantis Secure Registry and can be managed from the Mirantis Secure Registry GUI

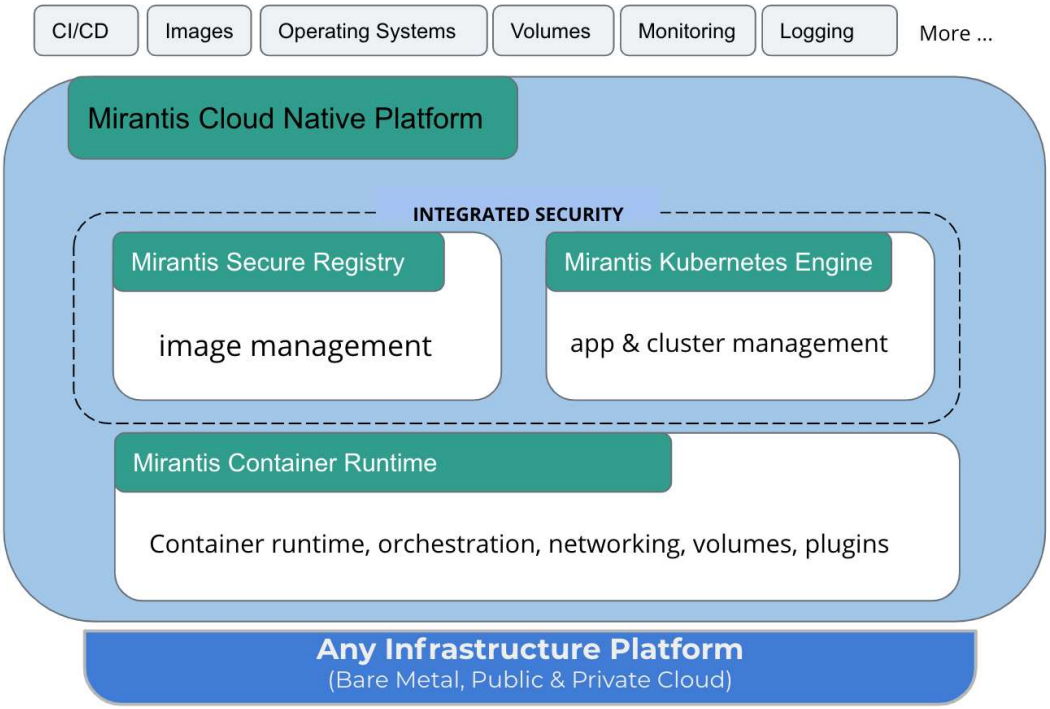

Figure 1 – Mirantis Cloud Native Platform Components

The Mirantis Crypto Library is a component of the Mirantis Container Runtime (versions 18.03 and later), and it supplies the cryptographic functionality necessary to support TLS-secured data and management communications between the Mirantis Container Runtime and the other Cloud Native Platform Cloud Native Platform components, cluster nodes, users, and external IT entities. It also supplies cryptographic functionality used to support Mirantis secrets, ID[10] hashes, encrypted overlay networks, and other Mirantis Cloud Native Platform components.

The Mirantis Crypto Library is validated at the FIPS 140-2 Section levels shown in Table 1 below.

Table 1 – Security Level per FIPS 140-2 Section

Section	Section Title	Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services, and Authentication	1

[10] ID - Identification

Section	Section Title	Level
4	Finite State Model	1
5	Physical Security	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC[11]	1
9	Self-tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	N/A

2.2 Module Specification

The Mirantis Crypto Library is a software module with a multiple-chip standalone embodiment. The overall security level of the module is 1.

The module includes a C-language cryptographic library based on the OpenSSL FIPS Canister (OpenSSL FIPS Object Module (FOM) version 2.0.14). The FOM is compiled into object form, fipscanister.o for the Linux-based systems, and then statically linked to an instance of the OpenSSL version 1.0.2k libcrypto library at build-time. The OpenSSL libcrypto library is statically linked to the GoCrypto Library[12], and then the GoCrypto Library, including the OpenSSL Library, is dynamically linked to the calling application, which is loaded into memory for execution by the operating system loader.

The module also includes a Go-language FIPS mode loader package. This pre-compiled loader package contains an init() function that acts as the module's default entry point, setting the FIPS mode and calling the FIPS self-tests automatically whenever the module is loaded into memory for execution, prior to the application package taking control of the process. This package also contains instructions allowing the Go-language applications to invoke the library APIs.

For FIPS 140-2 conformance testing, the module was tested and found compliant when running on the following environments:

- HPE[13] DL380 Gen9 with dual Intel Xeon E5-2670v3 processors running RHEL[14] v7.3 OS[15]
- HPE DL380 Gen9 with dual Intel Xeon E5-2670v3 processors running CentOS[16] v7.3 OS

The module implements the FIPS-Approved algorithms listed in Table 2 debajo de.

[11] EMI/EMC – Electromagnetic Interference / Electromagnetic Compatibility
[12] The GoCrypto Library is a crypto engine that supports non-FIPS validated crypto functions. It is out of scope for this validation.
[13] HPE – Hewlett Packard Enterprise
[14] LTS – Red Hat Enterprise Linux
[15] OS – Operating System
[16] CentOS – Community Enterprise Operating System

Table 2 – FIPS-Approved Cryptographic Algorithms

Certificate Number	Algorithm	Standard	Mode / Method	Key Lengths / Curves / Moduli	Use
5285, 5286	AES[17]	FIPS PUB 197	ECB[18], CBC[19], CTR[20], CFB1[21], CFB8, CFB128, OFB[22]	128, 192, 256	encryption/decryption
		NIST SP 800-38C	CCM[23]	128, 192, 256	encryption/decryption
		NIST SP 800-38D	GCM[24]	128, 192, 256	encryption/decryption
Vendor Affirmed	CKG[25]	NIST SP 800-133	-	-	key generation
1747, 1748	CVL Partial EC-DH	NIST SP 800-56A	ECC[26]	B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	shared secret computation
2031, 2032	DRBG[27]	NIST SP 800-90A	CTR-based	128, 192, 256	deterministic random bit generation
			HASH-based	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	deterministic random bit generation
			HMAC-based	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	deterministic random bit generation
1379, 1380	ECDSA[28]	FIPS PUB 186-4	Key Pair Generation, Public Key Verification	Key Pair Generation: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 Public Key Verification: B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521	key pair generation and public key verification
		FIPS PUB 186-4	SigGen, SigVer	SigGen: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P521 SigGen: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521	digital signature generation and verification

[17] AES – Advance Encryption Standard
[18] ECB – Electronic Code Book
[19] CBC – Cipher Block Chaining
[20] CTR – Counter
[21] CFB – Cipher Feedback
[22] OFB – Output Feedback
[23] CCM – Counter with CBC-MAC
[24] GCM – Galois Counter Mode
[25] CKG – Cryptographic Key Generation
[26] ECC – Elliptic Curve Cryptography
[27] DBRG – Deterministic Random Bit Generator
[28] ECDSA – Elliptic Curve Digital Signature Algorithm

Certificate Number	Algorithm	Standard	Mode / Method	Key Lengths / Curves / Moduli	Use
3492, 3493	HMAC[29]	FIPS PUB 198-1	SHA[30]-1, SHA-224, SHA-256, SHA-384, SHA-512	160, 224, 256, 384, 512	message authentication
2823, 2824	RSA[31]	FIPS PUB 186-4	SigGen9.31, SigGenPKCS[32]1.5, SigGenPSS[33]	2048, 3072, 4096-bit key sizes with SHA-1, SHA-224 (PKCS and PSS only), SHA-256, SHA-384, SHA-512	digital signature generation
			SigVer9.31, SigVerPKCS1.5, SigVerPSS	1024, 2048, 3072-bit key sizes with SHA-1, SHA-224 (PKCS and PSS only), SHA-256, SHA-384, SHA-512	digital signature verification
4244, 4245	SHS[11]	FIPS PUB 180-4	SHA-1, SHA-224, SHA-256. SHA-384, SHA-512	-	message digest
2670, 2671	Triple-DES[34]	NIST SP 800-67	TECB, TCBC, TCFB1, TCFB8, TCFB64, TOFB	Keying option 1	encryption/decryption

The module also employs the following non-Approved algorithm:

- Non-Deterministic Random Number Generator (NDRNG), for supplying entropy to the module's DRBG

The module uses a FIPS-Approved DRBG specified in NIST SP 800-90A to generate cryptographic keys. The resulting symmetric key or generated seed is an unmodified output from the DRBG. The DRBG requests entropy via a GET request to `/dev/random`, an NDRNG provided by the module's operational environment.

As a software module, the Mirantis Crypto Library has both a logical cryptographic boundary and a physical cryptographic boundary. The physical and logical boundaries are described in Sections 2.2.1 and 2.2.2, respectively.

2.2.1 Physical Cryptographic Boundary

As a software cryptographic module, the module has no physical components. Therefore, the physical boundary of the cryptographic module is defined by the hard enclosure around the host server on which it runs.

The host server hardware consists of a motherboard, a Central Processing Unit (CPU), random access memory (RAM), read-only memory (ROM), hard disk(s), hardware case, power supply, and fans. Other devices may be attached to the hardware appliance such as a monitor, keyboard, mouse, DVD[35] drive, printer, video adapter, audio adapter, or network adapter. In the validated configuration, the processor is an Intel Xeon processor. Please see Figure 2 for the host server block diagram and physical cryptographic boundary.

[29] HMAC – (Keyed-) Hashed Message Authentication Code
[30] SHA – Secure Shell
[31] RSA – Rivest Shamir Adleman
[32] PKCS – Public Key Cryptography Standard
[33] PSS – Probabilistic Signature Scheme
[11] SHS – Secure Hash Standard
[34] DES – Data Encryption Standard
[35] DVD – Digital Versatile Disc

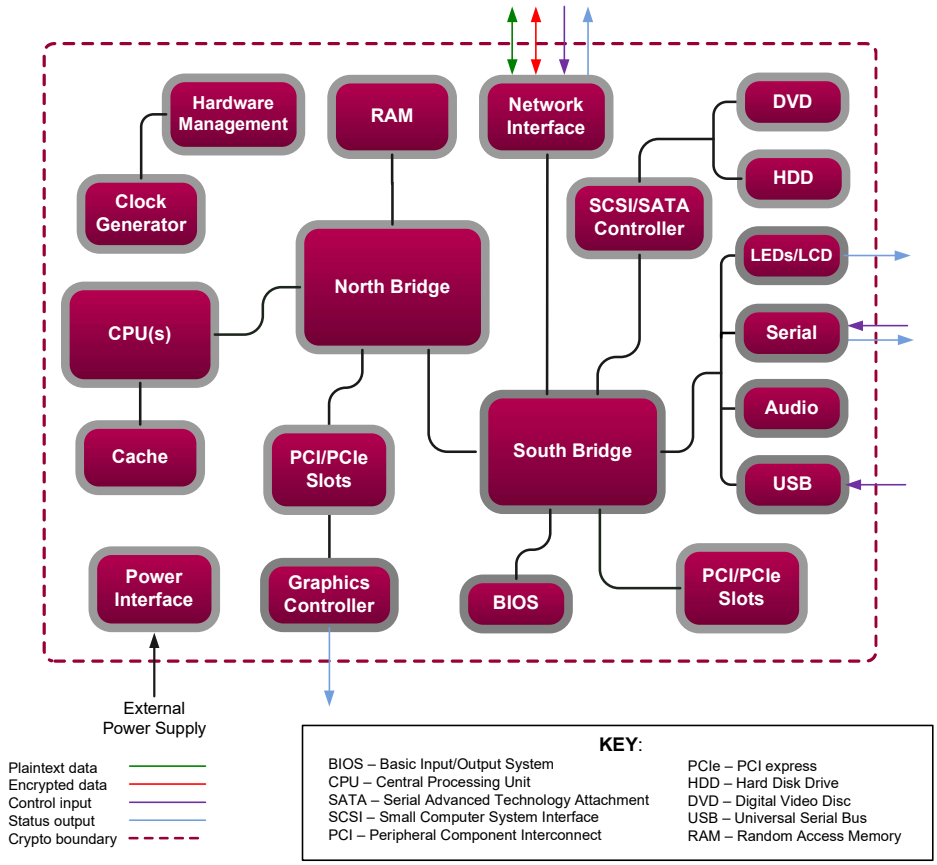

Figure 2 – Host Server Physical Block Diagram

2.2.2 Logical Cryptographic Boundary

The module is used by the calling application to provide symmetric cipher operation, digital signature generation and verification, hashing, cryptographic key generation, random number generation, and message authentication functions. The module is entirely contained within the physical cryptographic boundary described in Section 2.2.1.

Figure 3 shows the logical block diagram of the module executing in memory and its interactions with surrounding software components, as well as the module's logical cryptographic boundary.

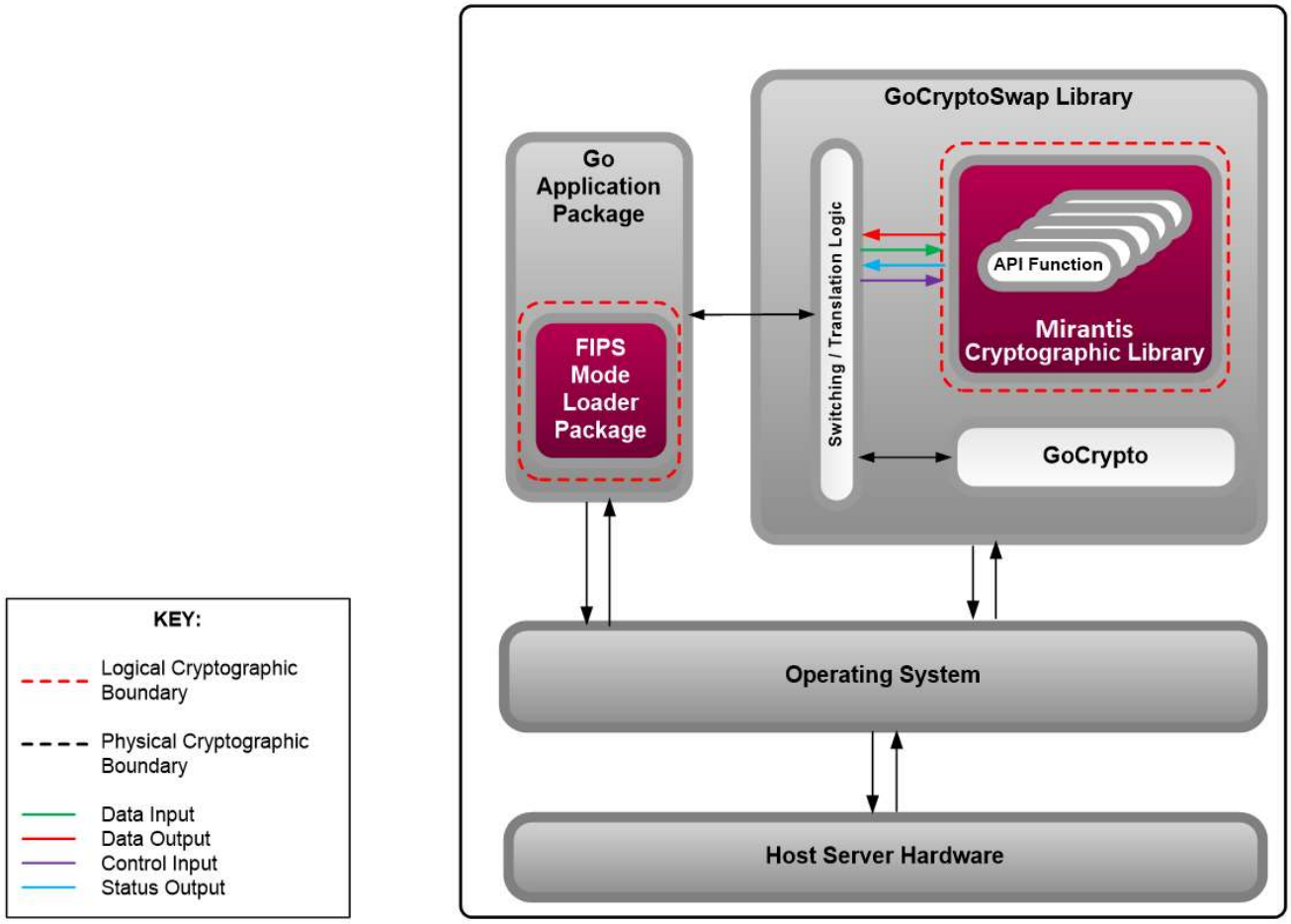

Figure 3 – Logical Block Diagram

2.3 Module Interfaces

The module isolates communications to logical interfaces that are defined in the software as an API[36]. The API interface is mapped to the following four logical interfaces:

- Data Input
- Data Output
- Control Input
- Status Output

The module's physical boundary features the physical ports of a host server. The module's manual controls, physical indicators, and physical, logical, and electrical characteristics are those of the host server. The module's logical interfaces are at a lower level in the software. The physical data and control input through physical mechanisms is translated into the logical data and control inputs for the software module. A mapping of the FIPS 140-2 logical interfaces, the physical interfaces, and the module interfaces can be found in Table 3.

[36] API – Application Programming Interface

Table 3 – FIPS 140-2 Logical Interface Mappings

FIPS Interface	Physical Interface	Module Interface (API)
Data Input	USB[37] ports (keyboard, mouse, data), network interface, serial ports	The API calls that accept input data for processing through their arguments.
Data Output	Graphics controller, USB ports, network interface, serial ports	The API calls that return, by means of their return codes or arguments, generated or processed data back to the caller.
Control Input	USB ports (keyboard, mouse), network interface, serial ports	The API calls that are used to initialize and control the operation of the module.
Status Output	Graphic controller, network interface, serial ports, Audio ports, LCDs[38]/LEDs[39]	Return values for API calls.
Power Input	AC[40] power socket	-

NOTE: As a software module, control of the physical ports is outside the scope of the module.

2.4 Roles and Services

There are two authorized roles that module operators may assume: Crypto Officer (CO) role and a User role. Since no authentication mechanisms are implemented, roles are assumed implicitly. When invoking a module service, a module operator implicitly assumes the CO and User roles simultaneously; thus, both roles have access to all module services. The module does not allow multiple concurrent operators in the FIPS-Approved mode of operation. As per section 6.1 of the NIST FIPS 140-2 Implementation Guidance, the calling application that loaded the module is the only operator.

Descriptions of the services available are provided in Table 4 below. Please note that the keys and Critical Security Parameters (CSPs) listed in the table indicate the type of access required using the following notation:

- R – Read: The CSP is read.
- W – Write: The CSP is established, generated, modified, or zeroized.
- X – Execute: The CSP is used within an Approved or Allowed security function or authentication mechanism.

Table 4 – Operator Services

Service	Operator		Description	CSP and Type of Access
	CO	User		
Initialize	✓	✓	Perform initialization of the module	None
Run self-test on demand	✓	✓	Performs power-up self-tests by power-cycling/rebooting the host server	None

[37] USB – Universal Serial Bus
[38] LCD – Liquid Crystal Display
[39] LED – Light Emitting Diode
[40] AC – Alternating Current

Service	Operator		Description	CSP and Type of Access
	CO	User		
Show status	✓	✓	Returns the current mode of the module	None
Zeroize	✓	✓	Zeroizes and de-allocates memory containing sensitive data	AES key – W Triple-DES key – W HMAC key – W RSA private/public key – W ECDSA private/public key – W EC-DH private/public component - W DRBG Seed – W DRBG Entropy – W DRBG 'Key' value - W DRBG 'C' value – W DRBG 'V' value – W
Generate random number	✓	✓	Returns the specified number of random bits to the calling application	DRBG Seed – WRX DRBG Entropy – RX DRBG 'Key' value - WRX DRBG 'C' value – WRX DRBG 'V' value – WRX
Generate message digest	✓	✓	Compute and return a message digest using SHS algorithms	None
Generate keyed hash (HMAC)	✓	✓	Compute and return a message authentication code	HMAC key – RX
Verify keyed hash (HMAC)	✓	✓	Verify a message authentication code	HMAC key – RX
Generate symmetric key	✓	✓	Generate and return the specified type of symmetric key (Triple-DES or AES)	AES key – W Triple-DES Key – W
Perform symmetric encryption	✓	✓	Encrypt plaintext using supplied key and algorithm specification (Triple-DES or AES)	AES key – RX Triple-DES key – RX
Perform symmetric decryption	✓	✓	Decrypt ciphertext using supplied key and algorithm specification (Triple-DES or AES)	AES key – RX Triple-DES key – RX
Perform authenticated symmetric encryption	✓	✓	Encrypt plaintext using supplied AES GCM key and IV	AES GCM key – RX AES GCM IV – RX
Perform authenticated symmetric decryption	✓	✓	Decrypt ciphertext using supplied AES GCM key and IV	AES GCM key – RX AES GCM IV – RX
Generate asymmetric key pair	✓	✓	Generate and return the specified type of asymmetric key pair (ECDSA)	ECDSA private/public key – W

Service	Operator		Description	CSP and Type of Access
	CO	User		
Calculate key agreement primitive	✓	✓	Calculate EC-DH key agreement primitive	EC-DH Public/Private components – WRX
Generate signature	✓	✓	Generate a signature for the supplied message using the specified key and algorithm (RSA or ECDSA)	RSA private key – RX ECDSA private key - RX
Verify signature	✓	✓	Verify the signature on the supplied message using the specified key and algorithm (RSA or ECDSA)	RSA public key – RX ECDSA public key - RX

2.5 Physical Security

The cryptographic module is a software module and does not include physical security mechanisms. Therefore, as per Section G.3 of the FIPS Implementation Guidance, requirements for physical security are not applicable.

2.6 Operational Environment

The module was tested and found to be compliant with FIPS 140-2 requirements on the following platforms and environments:

- HPE DL380 Gen9 with dual Intel Xeon E5-2670v3 processors running RHEL v7.3 OS
- HPE DL380 Gen9 with dual Intel Xeon E5-2670v3 processors running CentOS v7.3 OS

As per Section 6.1 of the *Implementation Guidance for FIPS PUB 140-2 and the CMVP*, the application that makes calls to the cryptographic module is the single user of the cryptographic module, even when the application is serving multiple clients.

All cryptographic keys and CSPs are under the control of the OS, which protects its CSPs against unauthorized disclosure, modification, and substitution. Additionally, the OS provides dedicated process space to each executing process, and the module operates entirely within the calling application's process space. The module only allows access to CSPs through its well-defined API.

2.7 Cryptographic Key Management

The module supports the CSPs listed debajo de in Table 5.

Table 5 – Cryptographic Keys, Cryptographic Key Components, and CSPs

CSP	CSP Type	Generation / Input	Output	Storage	Zeroization	Use
AES key	AES 128, 192, 256-bit key	Internally generated via Approved DRBG OR Input via API call parameter	Output in plaintext	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module; API call	Encryption, decryption
AES GCM key	AES GCM 128, 192, 256-bit key	Internally generated via Approved DRBG OR Input via API call parameter	Output in plaintext	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module; API call	Encryption, decryption
AES GCM initialization vector	Minimum 96-bit value	Internally generated via Approved DRBG (per SP 800-38D section 8.2.1)	Never output from the module	Plaintext in volatile memory	Power cycle/reboot; remove power; unload module; API call	Initialization vector for AES GCM
Triple-DES key	Triple-DES 168-bit key (Keying option 1)	Internally generated via Approved DRBG OR Input via API call parameter	Output in plaintext	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module; API call	Encryption, decryption
HMAC key	HMAC 160, 224, 256, 384, or 512 –bit key	Internally generated via Approved DRBG OR Input via API call parameter	Output in plaintext	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module; API call	Message authentication with SHS
ECDSA private key	Curves supported: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P521	Internally generated per FIPS 186-4 OR Input via API call parameter	Output in plaintext	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module; API call	Signature generation

CSP	CSP Type	Generation / Input	Output	Storage	Zeroization	Use
ECDSA public key	Curves supported: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521	Internally generated per FIPS 186-4 OR Input via API call parameter	Output in plaintext	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module; API call	Signature verification
EC-DH private component	Curves supported: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	Internally generated via Approved DRBG	Output in plaintext	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module; API call	Used by host application
EC-DH public component	Curves supported: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	Internally generated via Approved DRBG	Output in plaintext	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module; API call	Used by host application
RSA private key	RSA 2048, 3072, 4096-bit key	Input via API call parameter	Output in plaintext	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module; API call	Signature generation
RSA public key	RSA 1024, 2048, or 3072-bit key	Input via API call parameter	Output in plaintext	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module; API call	Signature verification
DRBG Seed	Random data – 440 or 880 bits	Internally generated using nonce along with DRBG entropy input.	Never output from the module	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module, API call	Seeding material for SP 800-90A DRBGs
DRBG Entropy	256-bit value	Externally generated and input via API call parameter	Never output from the module	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module, API call	Entropy material for SP 800-90A DRBGs
DRBG 'C' Value	Internal state value	Internally generated	Never output from the module	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module, API call	Used for Hash_DRBG
DRBG 'V' Value	Internal state value	Internally generated	Never output from the module	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module, API call r	Used for Hash_DRBG, HMAC_DRBG, and CTR_DRBG
DRBG 'Key' Value	Internal state value	Internally generated	Never output from the module	Keys are not persistently stored by the module	Power cycle/reboot; remove power; unload module, API call	Used for HMAC_DRBG and CTR_DRBG

2.8 EMI / EMC

The Mirantis Crypto Library was tested on the servers listed in Section 2.6 por encima de. These servers were tested and found conformant to the EMI/EMC requirements specified by 47 Code of Federal Regulations, Part 15, Subpart B, Unintentional Radiators, Digital Devices, Class A (business use).

2.9 Self-Tests

Cryptographic self-tests are performed by the module when the module is first powered up and loaded into memory as well as when a random number or asymmetric key pair is generated. The following sections list the self-tests performed by the module, their expected error status, and the error resolutions.

2.9.1 Power-Up Self-Tests

The module performs the following self-tests at power-up:

- Calling application integrity check (for FIPS Loader)
- OpenSSL FOM Software integrity check (using HMAC SHA-1)
- Known Answer Tests (KATs)
  - AES-ECB encrypt KAT
  - AES-ECB decrypt KAT
  - AES-CCM encrypt KAT
  - AES-CCM decrypt KAT
  - AES-GCM encrypt KAT
  - AES-GCM decrypt KAT
  - Triple-DES encrypt KAT
  - Triple-DES decrypt KAT
  - RSA KAT for sign/verify
  - HMAC KAT with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
  - CTR_DRBG KAT
  - Hash DRBG KAT
  - HMAC DRBG KAT
  - ECC Primitive "Z" Computation[41] test
- Pairwise Consistency Tests
  - ECDSA pairwise consistency test for sign/verify

Note: HMAC KATs with SHA-1 and SHA-2 utilize (and thus test) the full functionality of the SHA-1 and SHA-2 algorithms; thus, no independent KATs for SHA-1 and SHA-2 implementations are required.

2.9.2 Conditional Self-Tests

The module performs the following self-tests on power-up and conditionally:

[41] The ECC Primitive "Z" Computation KAT is compliant with FIPS 140-2 IG 9.6.

- Continuous RNG[42] Test for the DRBG
- Continuous RNG Test for the DRBG Entropy
- ECDSA pairwise consistency test for key pair generation

2.9.3 DRBG Health Checks

The module performs the following DRBG health checks on power-up and conditionally:

- SP 800-90 DRBG (Hash, HMAC, CTR) Instantiate Test
- SP 800-90 DRBG (Hash, HMAC, CTR) Generate Test
- SP 800-90 DRBG (Hash, HMAC, CTR) Reseed Test

2.9.4 Self-Test Failure Handling

If the module passes all the self-tests, it will return a value of "1" (indicating success) to the calling application, which indicates that the module is ready to be placed in its FIPS-Approved mode of operation.

If any of the above self-test fails, the self-test function returns a value of "0" (indicating failure). The module then ceases all cryptographic functionality and enters the critical error state. While in the critical error state, the module only returns "failure" status if any subsequent cryptographic services are requested. The module can only recover from the critical error state by rebooting the host server (thus restarting the module) and passing all power-on self-tests.

If rebooting the host server does not result in the successful execution of the self-tests, then the module will not be able to resume normal operations, and the CO should contact Mirantis, Inc.

2.10 Mitigation of Other Attacks

This section is not applicable. The module does not claim to mitigate any attacks beyond the FIPS 140-2 Level 1 requirements for this validation.

[42] RNG – Random Number Generator

3. Secure Operation

The sections below describe how to place and keep the module in the FIPS-Approved mode of operation. **Operating the module without following the guidance herein (including the use of undocumented services) will result in non-compliant behavior and is outside the scope of this Security Policy**.

3.1 Secure Management

The following paragraphs describe the steps necessary to ensure that the Mirantis Crypto Library is running in its FIPS-Approved manner.

3.1.1 Application Setup

The Mirantis Crypto Library is designed to provide cryptographic functionality to Go-language applications which are deployed in "package" form as part of Mirantis' family of CaaS solutions. Thus, while the module itself requires no installation or setup, the Go-language application packages employing the module must be properly configured.

The module includes C-language cryptographic library and a Go-language FIPS mode loader package. This loader package comprises a file called 'openssl.go' which contains an `init()` function that acts as the module's default entry point. The cgo[43] foreign function interface (FFI) is used in conjunction with the loader package in order for the module's C-language API to be accessible from Go-language calling applications.

The pre-compiled loader package is first present or linked somewhere in the `$GOPATH/src/` or `$GOROOTsrc/` tree. Then, in order to import the loader package and library API, the Go application package includes the following `import` instructions:

```
// #cgo pkg-config: openssl
import "C"
import _ "openssl"
```

The Go application package is then compiled and linked to the module library.

3.1.2 Configuration

At module load-time, the loader package's `init()` function automatically sets the module into FIPS mode and invokes the FIPS-required power-up self-tests prior to the calling application taking control of the execution process. Once all power-up self-tests have completed successfully, the module is running in its FIPS-Approved mode of operation. No additional configuration steps are required.

3.2 Operator Guidance

The following sections provide guidance to module operators for the correct and secure operation of the module.

[43] cgo enables the creation of Go packages that call C-language code.

3.2.1 Crypto Officer Guidance

No specific management activities are required to ensure that the module runs securely; once operational, the module only executes in a FIPS-Approved mode of operation. However, to verify the status, the Crypto Officer can use the `status()` function, which returns a mode indicator ("TRUE" when the module is operating in its Approved mode) and a version string.

If any irregular activity is observed or the module is consistently reporting errors, then Mirantis Customer Support should be contacted.

3.2.2 User Guidance

Although the User does not have any ability to modify the configuration of the module, they should notify the CO if any irregular activity is observed.

3.2.3 General Operator Guidance

The following provide further guidance for the general operation of the module:

- The module does not store any CSP persistently (beyond the lifetime of an API call), with the exception of DRBG state values used for the module's default key generation service. Zeroization of sensitive data is performed automatically by API function calls for temporarily stored CSPs.

- The module stores DRBG state values for the lifetime of the DRBG instance. The `FIPS_drbg_uninstantiate()` API can be used to explicitly destroy CSPs related to random number generation services.

- To determine the module's operational status, the `FIPS_mode()` API can be used. A non-zero return value indicates that the module is running in its FIPS-Approved mode; a "0" indicates non-FIPS mode.

- To execute the module's power-up self-tests on-demand, the module's host server can be rebooted/power-cycled.

3.3 Additional Guidance and Usage Policies

The notes below provide additional guidance and policies that must be followed by module operators:

- As a software cryptographic library, the module's services are intended to be provided to a calling application. Excluding the use of the NIST-defined elliptic curves as trusted third-party domain parameters, all other assurances from FIPS PUB 186-4 (including those required of the intended signatory and the signature verifier) are outside the scope of the module and are the responsibility of the calling application.

- The calling application shall ensure that the module's Triple DES, HMAC, and RSA algorithms are invoked using only key sizes that meet with their respective security strength requirements detailed in NIST SP 800-131A.

- The calling application is responsible for ensuring that the module performs no more than 2^{16} 64-bit data block encryptions under the same three-key Triple-DES key.

- The calling application shall use entropy sources that meet the security strength required for the DRBG as shown in SP 800-90A, Table 2 (Hash and HMAC DRBGs) and Table 3 (CTR DRBG). This entropy shall be supplied by means of a callback function. The callback function must return an error if the minimum entropy strength cannot be met.

- As the module does not persistently store keys, the calling application is responsible for the storage and zeroization of keys and CSPs passed into and out of the module.

- If power to the module is lost and subsequently restored, the calling application shall ensure that any AES-GCM keys used for encryption or decryption are re-distributed.

3.4 Non-FIPS-Approved Mode

When configured and operated per the guidance in this Security Policy, the module does not support a non-FIPS-Approved mode of operation.

4. Acronyms

Table 6 provides definitions for the acronyms used in this document.

Table 6 – Acronyms

Acronym	Definition
AC	Alternating Current
AES	Advanced Encryption Standard
API	Application Programming Interface
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC
CKG	Cryptographic Key Generation
CMVP	Cryptographic Module Validation Program
CO	Cryptographic Officer
CPU	Central Processing Unit
CSE	Communications Security Establishment
CSP	Critical Security Parameter
CTR	Counter
DES	Data Encryption Standard
DRBG	Deterministic Random Bit Generator
DVD	Digital Versatile Disc
ECB	Electronic Code Book
ECDSA	Elliptic Curve Digital Signature Algorithm
EMI/EMC	Electromagnetic Interference /Electromagnetic Compatibility
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
HMAC	(keyed-) Hash Message Authentication Code
HPE	Hewlett Packard Enterprise
ID	Identification
IT	Information Technology
KAS	Key Agreement Scheme
KAT	Known Answer Test
KPG	Key Pair Generation
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LTS	Long Term Support
NDRNG	Non-Deterministic Random Number Generator
NIST	National Institute of Standards and Technology

Acronym	Definition
OS	Operating System
PCIe	PCI express
PKCS	Public Key Cryptography Standard
PKV	Public Key Verification
PSS	Probabilistic Signature Scheme
RAM	Random Access Memory
RNG	Random Number Generator
RSA	Rivest, Shamir, and Adleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SP	Special Publication
TDES	Triple Data Encryption Standard
TLS	Transport Layer Security
USB	Universal Serial Bus

Prepared by:
Corsec Security, Inc.

13921 Park Center Road, Suite 460
Herndon, VA 20171
United States of America

Phone: +1 703 267 6050
Email: info@corsec.com
http://www.corsec.com