Huawei EulerOS 2.0 OpenSSH Server Cryptographic Module

Non-Proprietary FIPS 140-2 Security Policy
(Software Version 1.1)

Security Policy version: 1.3

Date: May 27, 2022

Table of Contents

List of Tables

List of Figures

1 Cryptographic Module Specification

This document is the non-proprietary FIPS 140-2 (Federal Information Processing Standards Publication 140-2) Security Policy for the Huawei EulerOS 2.0 OpenSSH Server Cryptographic Module, version 1.1 (version 8.2p1). It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS PUB 140-2 for a Security Level 1 module.

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

1.1 Module Overview

The Huawei EulerOS 2.0 OpenSSH Server Cryptographic Module (also referred to as “the module”) is a server daemon implementing the Secure Shell (SSH) protocol in the EulerOS 2.0 Operating System user space. The module interacts with other entities acting as SSH clients via the SSH protocol. The module only supports SSHv2 protocol.

The module uses the Huawei EulerOS 2.0 OpenSSL Cryptographic Module as a bound module (also referred to as “the bound OpenSSL module”), which provides the underlying cryptographic algorithms necessary for establishing and maintaining SSH sessions. The Huawei EulerOS 2.0 OpenSSL Cryptographic Module is a FIPS validated module (certificate #4235) in the same Operational Environments.

For the purpose of the FIPS 140-2 validation, the module is a software-only, multi-chip standalone cryptographic module validated at overall security level 1. The table below shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2 standard:

Table 1 - Security Level of Security Requirements

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

1.2 Module Specification

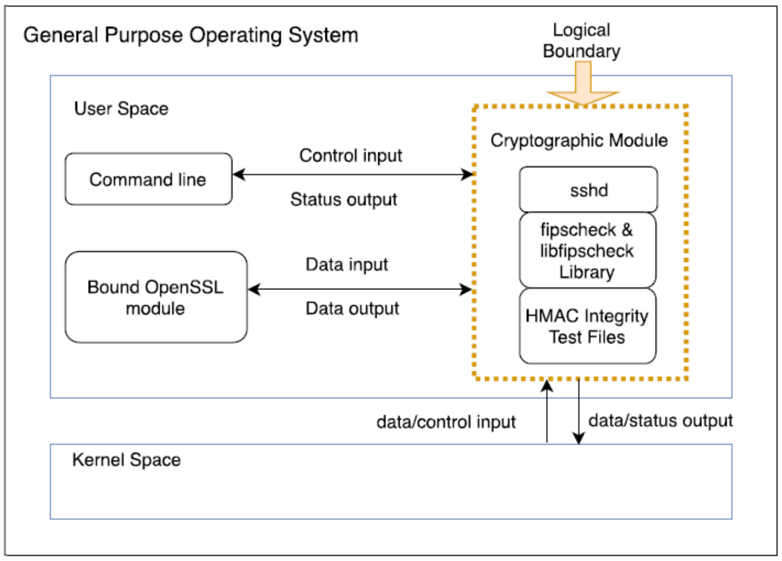
The logical cryptographic boundary of the module consists of the application, library files and their integrity test HMAC files, which are listed in Table 2.

Table 2– Cryptographic Module Components

Components	Description
/usr/sbin/sshd	SSH Server daemon
/usr/lib64/fipscheck/sshd.hmac	Integrity verification file for SSH Server daemon
/usr/bin/fipscheck	Cryptographic binary used to check the integrity
/usr/lib64/fipscheck/fipscheck.hmac	Integrity verification file for fipscheck binary
/usr/lib64/libfipscheck.so.1.2.1	Cryptographic library used to check the integrity
/usr/lib64/fipscheck/libfipscheck.so.1.2.1.hmac	Integrity verification file for libfipscheck.so.1.2.1 library

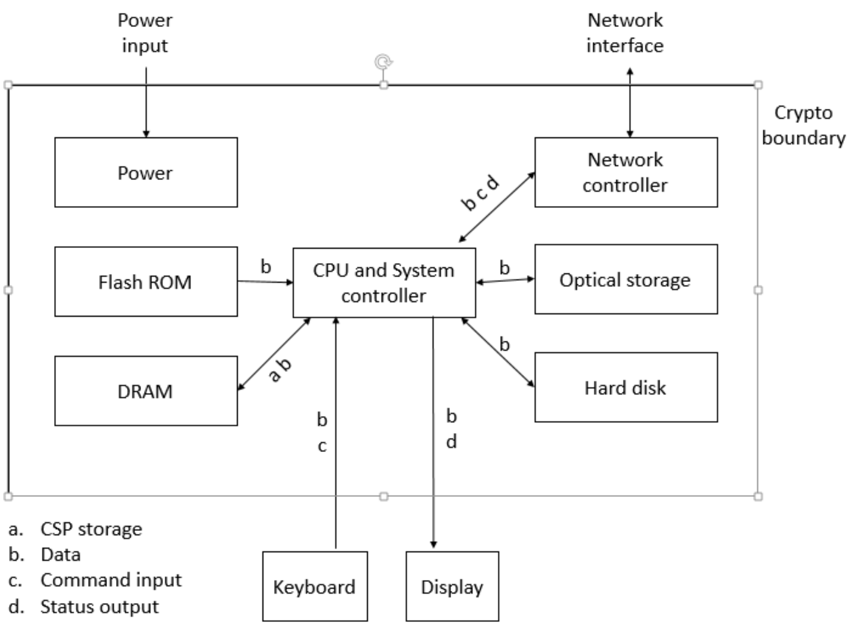
Figure 1 shows the logical block diagram of the module executing in memory on the host system. The logical cryptographic boundary is indicated with a dashed colored box.

Figure 1 – Logical Cryptographic Boundary



The module is aimed to run on a general purpose computer (GPC); the physical boundary is the surface of the case of the tested platforms, as shown in the diagram below:

Figure 2 – Cryptographic Module Physical Boundary



The module has been tested on the following platforms shown below:

Table 3 – Tested platforms

HW platform	OS & Version	Processor
Taishan200	Huawei EulerOS 2.0	Huawei Kunpeng 920
FusionServer RH2288	Huawei EulerOS 2.0	Intel Xeon E5-2690 v3

1.3 Algorithm implementation

The module implements the SSH KDF algorithm. The rest of the cryptographic algorithms are from the bound OpenSSL module. The cryptographic algorithms that are approved to be used in the FIPS mode of operation are tested and validated by the CAVP.

Table 4 – FIPS Approved Algorithms

Certificate Number	Algorithm	Standard	Mode/Method	Key Lengths Curves/module (in bits)	Use
#A1079	CVL (SSH)	[NIST SP 800-135]	SHA-1, SHA2-256, SHA2-384, SHA2-512	AES-128, AES-192, AES-256, TDES	Key derivation in the SSHv2 protocol

The following table shows Approved or allowed security functions provided by the bound OpenSSL module. There are algorithms, modes, and keys that have been CAVP tested but not used by the module. Only the algorithms, modes/methods, and key lengths/curves/moduli shown in this table are used by the module.

Table 5 – Algorithms from the bound OpenSSL Module

Certificate Number	Algorithm	Standard	Mode/Method	Key Lengths Curves/modulus (in bits)	Use
#A903	AES	[FIPS PUB 197]	CBC, CTR	128, 192, 256	Encryption/Decryption
		[NIST SP 800-38D]	GCM	128, 256	
	DRBG	[NIST SP 800-90A]	CTR-based	256	Deterministic Random Bit Generation
	DSA	[FIPS 186-4]	SHA-1	L:2048 – N:224 L:2048 – N:256 L:3072 – N:256	Signature verification
	ECDSA	[FIPS 186-4]	SHA-256, SHA-384, SHA-512	P-224, P-256, P-384, P-521	Key Generation Signature generation Signature verification
	HMAC	[FIPS PUB 198-1]	SHA-1, SHA2-256, SHA2-512	-	Integrity verification
	KAS–ECC-SSC	[NIST SP 800-56ARev3]	Ephemeral Unified scheme	P-224, P-256, P-384, P-521	EC Diffie-Hellman Key Agreement; Key establishment methodology provides between 112 and 256 bits of encryption strength.
	KAS–FFC-SSC	[NIST SP 800-56ARev3]	dhEphem scheme	P MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Diffie-Hellman Key Agreement; Key establishment methodology provides between 112 and 200 bits of encryption strength.
	KTS	FIPS PUB 197 NIST SP 800-38F FIPS PUB 198-1	TDES+HMAC	192	Key wrapping
	KTS	FIPS PUB 197 NIST SP 800-38F FIPS PUB 198-1	AES+HMAC	128, 192, 256	Key wrapping
	Safe Prime	[NIST SP 800-56ARev3]	Key Generation	MODP-2048,	Generation keys with SafePrimes groups

				MODP-3072, MODP-4096, MODP-6144, MODP-8192	
	RSA	[FIPS PUB 186-4]	PKCS 1.5	2048, 3072, 4096	Signature generation Signature verification
	SHS	[FIPS PUB 180-4]	SHA-1, SHA2-256, SHA2-384, SHA2-512	-	Message digest
	Triple-DES (TDES)	[SP 800-67][SP800-38A]	CBC	192	Encryption/Decryption
N/A	ENT (NP)	[NIST SP 800-90B]	-	-	Entropy source

The following table shows the non-approved functions provided by the bound OpenSSL module.

Table 6 – Non-Approved Cryptographic Functions

Algorithm	Usage
RSA signature generation with SHA-1 or with key sizes smaller than 2048 RSA signature verification with key sizes smaller than 1024 bits	Signature generation Signature verification
DSA signature generation with SHA-1 and 1024 bits key	Signature generation

1.4 Modes of Operation

The module supports two modes of operation:

1. **FIPS mode** (the Approved mode of operation): only approved or allowed security functions with sufficient security strength can be used.
2. **Non-FIPS mode** (the non-Approved mode of operation): The module is in non-FIPS mode when the non-Approved services are exercised.

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

The status of the module can be determined by the availability of the module. If the module is available, then it had passed all self-tests. If the module is unavailable, it is because the self-test failed and the module has transitioned to the error state

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

2 Cryptographic Module Ports and Interfaces

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

The logical interfaces are the sshd command, the messages sent to and received from the SSH client (SSHv2 protocol), and the application program interface (API) provided by the bound OpenSSL module. The following table summarizes the four logical interfaces:

Table 7 – Ports and Interfaces

Logical Interface Type	Port	Description
Data input	Keyboard, Ethernet port	Input parameters of the the sshd command through the command line with the host key files in /etc/ssh directory, /.ssh/authorized_keys, input data received via the SSHv2 channel, input data received via local or remote port-forwarding port, input data received from the bound OpenSSL module via its API parameters.
Data output	Display, Ethernet ports	Output data returned by the sshd command, output data sent via the SSHv2 channel, output data sent via local or remote port-forwarding port, output data sent to the bound OpenSSL module via its API parameters.
Control input	Keyboard, Ethernet ports	Invocation of the sshd command on the command line, control parameters via the sshd command or via the /etc/ssh/sshd_config file, SSHv2 protocol message requests received from SSH client.
Status output	Display, Ethernet ports	Status messages returned after execution of sshd command, status of processing SSHv2 protocol message requests.
Power input	GPC Power Supply Port	N/A

3 Roles, Services and Authentication

3.1 Roles

The module supports two distinct operator roles, User and Cryptographic Officer (CO). The details are in Table 9. The User and Crypto Officer roles are implicitly assumed by the entity accessing services implemented by the module.

Table 8 – Roles Description

Role ID	Role Description
Cryptographic Officer (CO)	Performs services of configuration and terminate sshd application.
User	Performs services to establish, maintain and close SSH session, show status and self-tests

3.2 Services

The module provides services to users that assume one of the available roles. All services implemented by the module are listed in tables below. The approved services are shown in Table 10 and the non-approved but allowed services available in FIPS Approved mode are shown in Table 11. Each service description also describes all usage of CSPs by the service.

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

Table 9 – Services in FIPS mode

Service	Role	Description	Input	Output	CSP and Type of Access
Establish SSH Session	User	SSH authentication	Certificate	None	RSA Server Public Key – R/X RSA Server Private Key – R ECDSA Server Public Key – R/X ECDSA Server Private Key – R RSA Client Public Key – R ECDSA Client Public Key – R DSA Client Public Key – R

Service	Role	Description	Input	Output	CSP and Type of Access
		Negotiate a SSH V2 key agreement	None	shared secret	Server Diffie-Hellman public key - W Server Diffie-Hellman private key – W/X Server EC Diffie-Hellman public key – W Server EC Diffie-Hellman private key – W/X Client Diffie-Hellman public key – R/X Client EC Diffie-Hellman public key – R/X
		Key derivation using SP800-135 SSH KDF	None	Session key	Diffie-Hellman Shared Secret - W/R/X EC Diffie-Hellman Shared Secret - W/R/X Triple-DES Session keys – W/X AES Session keys – W/X Session data authentication keys (HMAC) – W/X
Maintain SSH Session	User	Provide data encryption/decryption and data authentication over SSH V2 network protocol	None	None	Triple-DES Session keys – X AES Session keys – X Session data authentication keys (HMAC) – X
Close SSH Session	User	Zeroize SSH derived session encryption and data authentication keys by closing the SSH session	Command Line	None	Triple-DES Session keys – W AES Session keys – W Session data authentication keys (HMAC) – W
Terminate sshd application	CO	Zeroize SSH derived session encryption and data authentication keys by terminating the sshd application	Command Line	None	Triple-DES Session keys – W AES Session keys – W Session data authentication keys (HMAC) – W
Configure SSH Server	CO	Configure the SSH Server	Modifying /etc/ssh/sshd_config	None	N/A
Show status	User	Show status of the module	Command Line	status	N/A

Service	Role	Description	Input	Output	CSP and Type of Access
Self-test	User	Perform on-demand self-tests	None	None	N/A

Table 10– Services in non-FIPS mode

Service	Role	Description	Input	Output	CSP and Type of Access
Establish SSH Session	User	SSH authentication (Digital Signature using non-Approved algorithms or key sizes. • DSA signature generation with SHA-1 and 1024-bit key. • RSA signature generation with SHA-1 and/or key sizes smaller than 2048 bits, RSA signature verification with key sizes smaller than 1024 bits.)	Certificate	None	RSA Server Public Key – R/X RSA Server Private Key – R DSA Server Public Key – R/X DSA Server Private Key – R RSA Client Public Key – R DSA Client Public Key – R

3.3 Authentication

The module does not support operator authentication mechanisms. The user roles are implicitly assumed based on the service requested.

4 Physical Security

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

5 Operational Environment

The module operates in a modifiable operational environment per FIPS 140-2 level 1 specifications. The module runs on the operating system executing on the hardware platforms listed in Table 3.

The operating system is restricted to a single operator. Concurrent operators are explicitly excluded. The application that requests cryptographic services is the single user of the module.

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 process space.

6 Cryptographic Key Management

All Critical Security Parameters (CSPs) and how they are used and managed by the module are described in this section.

6.1 Critical Security Parameters

The following table summarizes all the CSPS in details.

Table 11– Critical Security Parameters (CSPs)

CSP	CSP Type	Generation/Input	Output	Storage	Zeroization	Use
RSA Server Private Key	2048, 3072, 4096	Read from the host key file	The keys are output to the bound OpenSSL module via API parameters	Not persistently stored	Zeroized automatically when closing SSH session or terminating sshd application	Signature generation
RSA Server Public Key	2048, 3072, 4096	Read from the host key file	The keys are output during SSH session handshake	Not persistently stored		Signature verification
ECDSA Server Private Key	P-256, P-384, P-521	Read from the host key file	The keys are output to the bound OpenSSL module via API parameters	Not persistently stored		Signature generation
ECDSA Server Public Key	P-256, P-384, P-521	Read from the host key file	The keys are output during SSH session handshake	Not persistently stored		Signature verification
ECDSA Client Public Key	P-256, P-384, P-521	Entered during SSH authentication	The keys are output to the bound OpenSSL module via API parameters	Not persistently stored		Signature verification
RSA Client Public Key	2048, 3072, 4096	Entered during SSH authentication	The keys are output to the bound OpenSSL module via API parameters	Not persistently stored		Signature verification
Server Diffie-Hellman public key	p=2048, q=224; p=2048, q=256	Entered from the bound OpenSSL module via API parameters	The keys are output SSH session handshake	Not persistently stored		Derive shared secret
Server EC Diffie-Hellman public key	P-256, P-384, P-521	Entered from the bound OpenSSL module via API parameters	The keys are output SSH session handshake	Not persistently stored		Derive shared secret

CSP	CSP Type	Generation/Input	Output	Storage	Zeroization	Use
Server Diffie-Hellman private key	p=2048, q=224; p=2048, q=256	Entered from the bound OpenSSL module via API parameters	The keys are output to the bound OpenSSL module via API parameters	Not persistently stored		Derive shared secret
Server EC Diffie-Hellman private key	P-256, P-384, P-521	Entered from the bound OpenSSL module via API parameters	The keys are output to the bound OpenSSL module via API parameters	Not persistently stored		Derive shared secret
Client Diffie-Hellman public key	p=2048, q=224; p=2048, q=256	Entered during SSH handshake	The keys are output to the bound OpenSSL module via API parameters	Not persistently stored		Derive shared secret
Client EC Diffie-Hellman public key	P-256, P-384, P-521	Entered during SSH handshake	The keys are output to the bound OpenSSL module via API parameters	Not persistently stored		Derive shared secret
Diffie-Hellman Shared Secret	DH derived keys	Entered from the bound OpenSSL module via API parameters	Never exits the module	Not persistently stored		SSH communication
EC Diffie-Hellman Shared Secret	ECDH derived keys	Entered from the bound OpenSSL module via API parameters	Never exits the module	Not persistently stored		SSH communication
AES Session keys	128, 192, 256	Derived from the shared secret via SP800-135 SSH KDF	The keys are output to the bound OpenSSL module via API parameters	Not persistently stored		Symmetric encryption
Triple-DES Session keys	192	Derived from the shared secret via SP800-135 SSH KDF	The keys are output to the bound OpenSSL module via API parameters	Not persistently stored		Symmetric encryption
Session data authentication keys (HMAC)	112 bits or greater	Derived from the shared secret via SP800- 135 SSH KDF	The keys are output to the bound OpenSSL module via API parameters	Not persistently stored		Keyed-Hash Message Authentication

6.2 Random Number Generation

The module does not implement any random number generator. Instead, it uses the Random Number Generation service provided by the bound OpenSSL module, which implements a Deterministic Random Bit Generator (DRBG) based on [SP800-90A].

6.3 Key Agreement

The module does not implement any Key agreement. Instead, it uses the Key Agreement services provided by the bound OpenSSL module, which implements Diffie-Hellman and EC Diffie-Hellman key agreement schemes.

6.4 Key Derivation

The module implements SP800-135 SSH KDF for the SSHv2 protocol, which is considered an approved key establishment method according the Annex D of the FIPS PUB 140-2 document.

6.5 Key Entry/Output

Keys are brought into the module via well-defined APIs in plain text, or generated internally (via SSH function KDF SP800-135rev2). Conversely, keys are output to the module in plain text.

The module does not support manual key entry or intermediate key generation key output.

6.6 Key/CSP Storage

The module does not perform persistent storage of keys. The keys and CSPs are temporarily stored as plaintext in the RAM.

The server public and private keys are stored in the host key files in /etc/ssh directory, which are within the module physical boundary but outside its logical boundary.

The HMAC key used for the Integrity Test is stored in the module and relies on the operating system for protection.

6.7 Key/CSP Zeroization

Zeroization occurs when the “Close SSH session” and the “Terminate sshd application” services are invoked.

The memory occupied by keys is allocated by regular memory allocation operating system calls. The module calls appropriate key zeroization functions provided by the bound OpenSSL module, which overwrite the memory occupied by keys with “zeros” and deallocate the memory with the regular memory deallocation operating system call.

6.8 Key Transport

The module provides approved key transport methods according to IG D.9 exclusively within the context of the SSH protocol. The methods are available once the SSH connection is established using the approved services of this module. The approved methods are provided with the assistance of the bound module by using a combination method, consisting of using an approved symmetric encryption mode from the bound module (e.g., AES, Triple-DES) together with an approved message authentication method from the bound module (e.g., HMAC) as follows:

- KTS (AES Cert. #A903 and HMAC Cert. #A903; key establishment methodology provides between 128 and 256 bits of encryption strength)
- KTS (T-DES Cert. #A903 and HMAC Cert. #A903; key establishment methodology provides 192 bits of encryption strength)

7 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The Huawei EulerOS 2.0 OpenSSH Server Cryptographic Module was tested on the servers listed above, in Table 3 – Tested Platforms. These servers were tested and found to conform to the EMI/EMC requirements specified by 47 Code of Federal Regulations, Part 15, Subpart B, Unintentional Radiators, Digital Devices, Class A (Business use).

8 Self-tests

8.1 Power Up Self-tests

At the beginning of the execution, the cryptographic module automatically performs the power-up self-tests to ensure that neither it and nor its components have been modified. Note that during the execution of the on-demand self-tests, crypto services are not available and no data output or input is possible.

The integrity of the module is verified by comparing an HMAC-SHA2-256 value calculated at run time with the HMAC value stored in the .hmac file that was computed at build time for each software component of the module. If the HMAC values do not match, the test fails and the module enters the error state.

The integrity verification is performed as follows: the OpenSSH Server application links with the library libfipscheck.so which is intended to execute “fipscheck” to verify the integrity of the OpenSSH server application file using the HMAC-SHA2-256 algorithm. Upon calling the FIPSCHECK_verify() function (it returns ‘1’ if the integrity test successes. Otherwise, it returns ‘0’ and prints ‘FIPS integrity verification test failed’ throughout the status output interface) provided with libfipscheck.so, fipscheck is loaded and executed, and the following steps are performed:

- The OpenSSL module loaded by “fipscheck”, performs the integrity check of the OpenSSL library files using the HMAC-SHA2-256 algorithm.
- “fipscheck” performs the integrity check of its application file and automatically verifies the integrity of “libfipscheck.so” before processing requests of calling applications.
- The “fipscheck” application performs the integrity check of the OpenSSH server application file. The fipscheck computes the HMAC-SHA2-256 checksum of that and compares the computed value with the value stored inside the /usr/lib64/fipscheck/<application filename>.hmac checksum file. The libfipscheck.so library reports the result to the OpenSSH server application.

8.2 Cryptographic Algorithm Tests

The module uses the Huawei EulerOS 2.0 OpenSSL Cryptographic Module as a bound module which provides the underlying cryptographic algorithms. All the known answer tests (KAT) are implemented by the bound OpenSSL module.

Per section 9.4 of Implementation Guidance document, no KAT KDF for SSH is required because this function is considered an algorithm component.

8.3 Conditional Self-Tests

The module does not perform conditional tests.

8.4 On-Demand self-tests

The module provides the Self-Test service to perform self-tests on demand. On demand self-tests can be invoked by powering-off and reloading the module. This service performs the same cryptographic algorithm tests executed during power-up. During the execution of the on-demand self-tests, crypto services are not available and no data output or input is possible.

9 Guidance

This section documents the guidance for the cryptographic officer and the user. In order to satisfy requirements in FIPS 140-2, the cryptographic officer and the user should follow this guidance to maintain proper use of the module.

9.1 Crypto Officer Guidance

9.1.1 Module Installation

The vendor provides the ISO file of the EulerOS 2.0 Operating System fully operational with the module ready to operate in FIPS mode. No more actions are needed (such as install someone else .rpm packet) by the operator to work with the Cryptographic Module.

Prior to the Operating System installation, the vendor encourages the operator to check the SHA-1 digest value of the "ISO" binaries.

- **ARM-based ISO**: EulerOS-V2.0SP9-aarch64-dvd.iso

  SHA-1: ***c0cc3041e77582dfedcb51110fbc855ea68a2aa5***

- **x86-based ISO**: EulerOS-V2.0SP9-x86_64-dvd.iso

  SHA-1: ***28a7c73b2d4dc69a973bc4f8a14815e679c0dd6f***

9.1.2 Configurations

For the module, the mode of operation is implicitly assumed depending on the services/security functions invoked as stated in section 3.2. Successive sections list the available ciphers from the module. Any use of non-approved cipher or non-Approved key size will result in the module entering the non-FIPS mode of operation. With operating environment setup as stated in the above section, the following restrictions are applicable. No cipher addition is possible by configuration or command line options.

1. SSH protocol version 1 is not allowed
2. GSSAPI is not allowed.
3. Only the following ciphers, Message authentication code algorithms as well as key exchange schemes are allowed:

- ***Ciphers:***
  - aes256-ctr
  - aes128-gcm@openssh.com
  - aes256-gcm@openssh.com
  - aes128-cbc
  - aes256-cbc

  Also configurable:

  - aes128-ctr
  - aes192-ctr
  - aes192-cbc
  - 3des-cbc

- ***KexAlgorithms:***
  - diffie-hellman-group14-sha256
  - diffie-hellman-group16-sha512
  - diffie-hellman-group18-sha512
  - diffie-hellman-group-exchange-sha256
  - ecdh-sha2-nistp256
  - ecdh-sha2-nistp384
  - ecdh-sha2-nistp521
- ***MACs:***
  - hmac-sha2-512
  - hmac-sha2-512-etm@openssh.com
  - hmac-sha2-256-etm@openssh.com
  - hmac-sha1
  - hmac-sha1-etm@openssh.com

9.2 User Guidance

This module is designed to be used by connecting it with an SSH client. To use a FIPS validated SSH client, please see the Huawei EulerOS 2.0 OpenSSH Client Cryptographic Module FIPS 140-2 Non-Proprietary Security Policy.

In order to run the module in FIPS Approved mode, the user can only use services and security functions listed in Table 4 and Table 5. The user shall ensure that the module is in FIPS mode during using the module.

9.2.1 OpenSSH Server Managment

To start manually the sshd daemon, use the following command:

- systemctl start sshd

To stop the sshd daemon, use the following command:

- systemctl stop sshd

To check the status of the sshd daemon, use the following command:

- systemctl status sshd

To operate the module in FIPS mode, please consider the following restrictions:

- Only the SSHv2 cipher suites listed in 9.1.2 section are available to be used.

- Use of 1024-bit DSA keys for signature generation will result in the module entering non-FIPS mode implicitly. The DSA signature verification with 1024-bit key is only for legacy use.

- Use of less than 2048-bit RSA keys for signature generation or less than 1024-bit RSA keys for signature verification will result in the module entering non-FIPS mode implicitly.

9.2.2 SSH

No parts of the SSH protocol have been tested by the CAVP or CMVP, but for the key derivation function (KDF).

9.2.3 Triple-DES

Data encryption using the same three-key Triple-DES key shall not exceed 2^{16} Triple-DES (64-bit) blocks, in accordance to [SP800-67] and IG A.13. The user of the module is responsible for ensuring the module's compliance with this requirement.

9.2.4 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 nonce_explicit part of the IV does not exhaust the maximum number of possible values for a given session key. The design of the SSH protocol in this module implicitly ensures that the nonce_explicit, or counter portion of the IV will not exhaust all of its possible values.

The AES GCM IV generation is in compliance with the [RFC5647] and shall only be used for the SSH protocol version V2 to be compliant with IG A.5, provision 1 (" SSHv2 protocol IV generation "); Moreover, the module is compliant with Section 3.3.1 of [SP800-52] Rev2."

9.2.5 Handling Self-Test Errors

When the self-test fails, the module returns the following message 'FIPS integrity verification test failed' and enters in the error state (Critical Error State). In error state, no cryptographic operation is available. The module must be restarted and perform power-up test again to recover from this error state.

10 Mitigation of Other Attacks

The module does not mitigate against attacks.

11 References and Definitions

Table 12 – References

Abbreviation	Full Specification Name
[FIPS 140-2]	FIPS 140-2 Security Requirements for Cryptographic modules
[IG]	Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program
[FIPS 197]	FIPS 197 Advanced Encryption Standard
[FIPS 180-4]	FIPS 180-4 Secure Hash Standard
[FIPS 198-1]	FIPS 198-1 The Keyed-Hash Message Authentication Code (HMAC)
[FIPS 186-4]	FIPS 186-4 Digital Signature Standard (DSS)
[SP 800-67]	NIST SP 800-67 Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher
[SP 800-38A]	NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation Methods and Techniques
[SP 800-38D]	NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
[SP 800-38F]	NIST SP 800-38F, - Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping
[SP 800-56A]	NIST SP 800-56A, Recommendation for Pair-Wise Key Establishment Schemes using Discrete Logarithm Cryptography , rev2
[SP 800-90A]	NIST SP 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators
[SP 800-131A]	NIST SP 800-131A, Transitioning the Use of Cryptographic Algorithms and Key Lengths
[SP 800-135]	NIST SP 800-135, Recommendation for Existing Application-Specific Key Derivation Functions, rev1

Table 13– Acronyms

Acronym	Definition
AES	Advanced Encryption Standard
API	Application Program Interface
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CMVP	Cryptographic Module Validation Program

Acronym	Definition
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
EMI/EMC	Electromagnetic Interference/Electromagnetic Compatibility
FIPS	Federal Information Processing Standards Publication
GCM	Galois Counter Mode
GPC	General Purpose Computer
HAMC	Hash Message Authentication Code
IG	Implementation Guidance
KAT	Known Answer Test
KDF	Key Derivation Function
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
PSS	Probabilistic Signature Scheme
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
TLS	Transport Layer Security