Huawei EulerOS 2.0 OpenSSL Cryptographic Module

Non-Proprietary FIPS 140-2 Security Policy

(Software Version 1.1)

Security Policy version: 2.5

Date: May 12, 2022

Table of Contents

List of Tables

List of Figures

1 Cryptographic Module Specification

This document is the non-proprietary FIPS 140-2 Security Policy of the Huawei EulerOS 2.0 OpenSSL Cryptographic Module, software version 1.1 (based on OpenSSL 1.1.1f-105). It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS PUB 140-2 (Federal Information Processing Standards Publication 140-2) for a Security Level 1 software module.

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

1.1 Module Overview

The Huawei EulerOS 2.0 OpenSSL Cryptographic Module (hereafter referred to as “the module”) is a software library supporting the implementation of Transport Layer Security (TLS) protocol v1.2 as well as general purpose approved cryptographic algorithms listed below.

The module provides cryptographic services to applications running in the user space of the underlying EulerOS 2.0 operating system through a C language application program interface (API).

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 following table shows the claimed security level 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	1

1.2 Module Specification

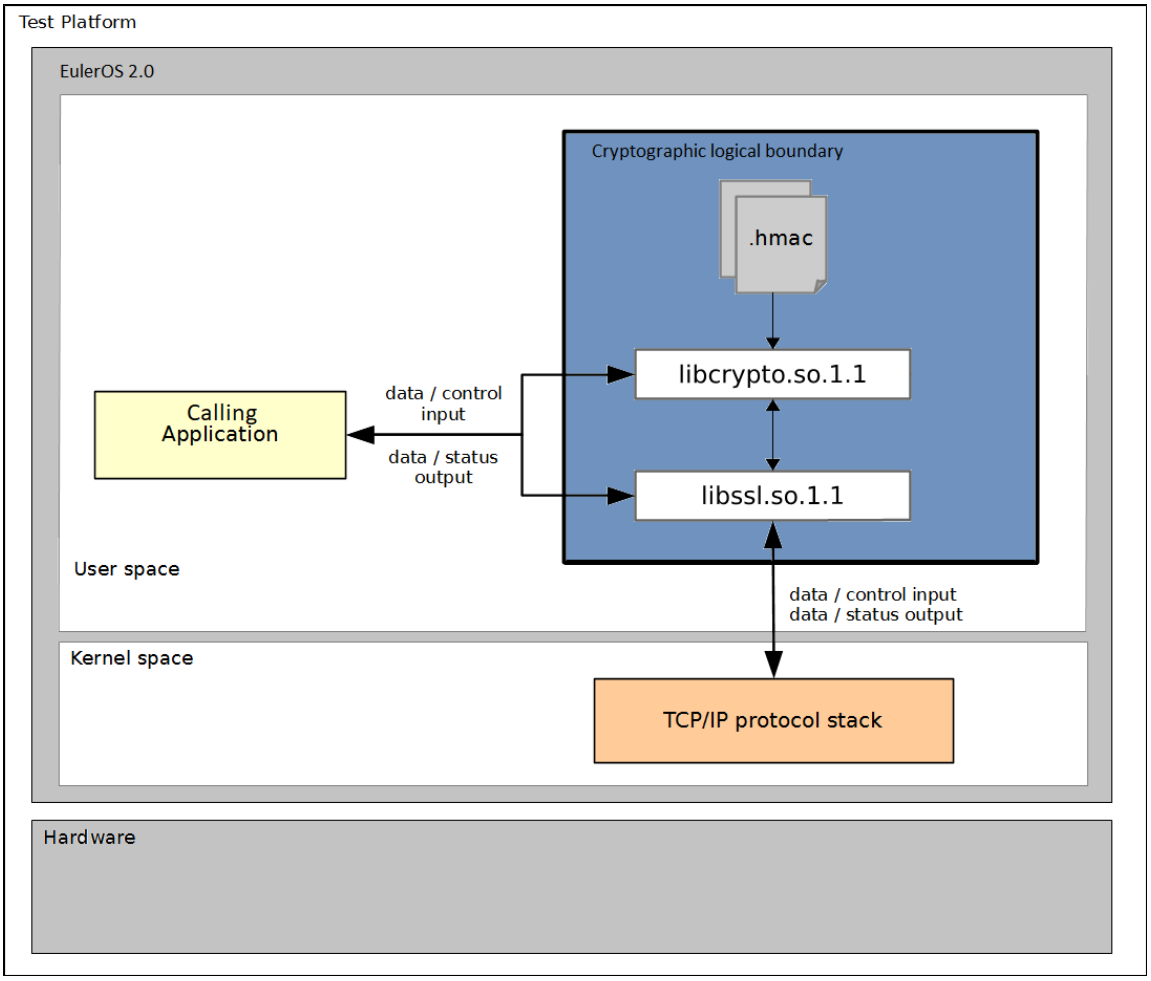
The cryptographic logical boundary consists of all shared libraries and the integrity check files used for Integrity Tests. The following table enumerates the files that comprise the module.

Table 2 – Cryptographic Module Components

Component	Description
libssl.so.1.1	Shared library for TLS network protocols
libcrypto.so.1.1	Shared library for cryptographic implementations
libssl.so.1.1.hmac	Integrity check signature for libssl shared library
libcrypto.so.1.1.hmac	Integrity check signature for libcrypto shared library

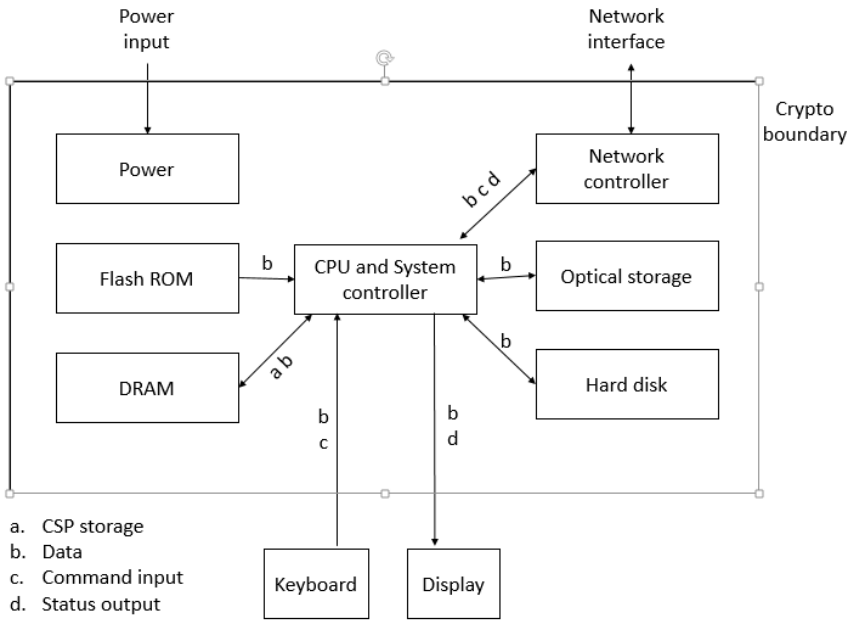
The software block diagram below shows the module, its interfaces with the operational environment and the delimitation of its logical boundary, comprised of all the components within the blue box.

Figure 1 – Cryptographic Boundary



The module is aimed to run on a general purpose computer (GPC); the physical boundary of the module is the tested platforms. Figure 2 shows the major components of a GPC.

Figure 2 – Cryptographic Module Physical Boundary



The module has been tested on the 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 E5 Xeon E5-2690

1.3 Algorithm implementation

The module supports the following FIPS Approved Algorithms listed in Table 4 below:

Table 4 – FIPS Approved Algorithms

Certificate Number	Algorithm	Standard	Mode/Method	Key Lengths Curves/module (in bits)	Use
#A903	AES	[FIPS PUB 197]	ECB, CBC, OFB, CFB1, CFB8, CFB128, CTR	128, 192, 256	Encryption/Decryption
		[NIST SP 800-38B]	CMAC	128, 192, 256	MAC Generation/Verification
		[NIST SP 800-38C]	CCM	128, 192, 256	Encryption/Decryption

		[NIST SP 800-38D]	GCM, GMAC	128, 192, 256	Encryption/Decryption MAC Generation/Verification
		[NIST SP 800-38E]	XTS	128, 256	Encryption/Decryption for data storage
		[NIST SP 800-38F]	KW	128, 192, 256	Key Wrapping and Unwrapping
	CVL	[NIST SP 800-135rev1]	TLS 1.2	-	Application-specific Key derivation
	DRBG	[NIST SP 800-90A]	CTR-based	256	Deterministic Random Bit Generation
	DSA	[FIPS 186-4]	pqgGen	L:2048 – N:224 L:2048 – N:256 L:3072 – N:256	Domain Parameters Generation
			pqgVer	L:2048 – N:224 L:2048 – N:256 L:3072 – N:256	Domain Parameters Verification
			keyGen	L:2048 – N:224 L:2048 – N:256 L:3072 – N:256	Key pair generation
			sigGen	L:2048 – N:224 L:2048 – N:256 L:3072 – N:256	Digital Signature Generation
			sigVer	L:2048 – N:224 L:2048 – N:256 L:3072 – N:256	Digital Signature Verification
	ECDSA	[FIPS 186-4]	keyGen	P-224, P-256, P-384, P-521	Key pair generation
			keyVer	P-224, P-256, P-384, P-521	Key pair verification
			SigGen	P-224, P-256, P-384, P-521	Digital Signature Generation
			SigVer	P-224, P-256, P-384, P-521	Digital Signature Verification
	HMAC	[FIPS PUB 198-1]	SHA-1, SHA2-224, SHA2-256,	-	Keyed-Hash Message Authentication Code

			SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512		
	KAS –ECC-SSC	[NIST SP 800-56ARev3]	Ephemeral Unified scheme	P-224, P-256, P-384, P-521	Shared Secret Computation
	KAS –FFC-SSC	[NIST SP 800-56ARev3]	dhEphem scheme	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Shared Secret Computation
	KAS	[NIST SP 800-56ARev3] in conjunction with [SP800-135]	Ephemeral Unified scheme	P-224, P-256, P-384, P-521	EC Diffie-Hellman Key Agreement; KAS-ECC-SSC in conjunction with KDF TLS (1.2) of SP800-135, which is compliant with the "Scenario X1, path 2" of the [IG]; Key establishment methodology provides between 112 and 256 bits of encryption strength.
	KAS	[NIST SP 800-56ARev3] in conjunction with [SP800-135]	dhEphem scheme	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Diffie-Hellman Key Agreement; KAS-FFC-SSC in conjunction with KDF TLS (1.2) of SP800-135, which is compliant with the "Scenario X1, path 2" of the [IG]; Key establishment methodology provides between 112 and 200 bits of encryption strength.
	Safe Prime	[NIST SP 800-56ARev3]	Key Generation and Verification	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Generation and verification of keys with SafePrimes groups

	RSA	[FIPS PUB 186-4]	KeyGen	2048, 3072	Key pair generation
			SigGen (ANSI X9.31, PKCS 1.5, PKCSPSS)	2048, 3072, 4096	Signature generation
			SigVer (ANSI X9.31, PKCS 1.5, PKCSPSS)	2048, 3072, 4096	Signature verification
	SHA-3	[FIPS PUB 202]	SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	-	Message digest
	SHS	[FIPS PUB 180-4]	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	-	Message digest
	Triple-DES (TDES)	[SP 800-67]	ECB, CBC, CFB1, CFB8, CFB64, OFB	192	Encryption/Decryption
		[SP800-38B]	CMAC	168	MAC Generation/Verification
-	ENT (NP)	[NIST SP 800-90B]	-	-	Non-deterministic random bit generation

The module also supports some non-approved cryptographic algorithms. Table 5 describes the non-Approved but allowed algorithms in FIPS mode, and Table 6 describes the non-approved algorithms. Note that these algorithms in Table 6 are only available in Non-FIPS mode.

Table 5 – Non-Approved but Allowed Cryptographic Functions

Algorithm	Caveat	Use
RSA Key Encapsulation with Encryption and Decryption Primitives with keys equal or larger than 2048 bits up to 15360 or more.	RSA (key wrapping; key establishment methodology provides between 112 and 256 bits of encryption strength)	Key Establishment; allowed per [IG D.9]
SHA-1 used in the Digital Signature Generation	-	According [SP800-52], SHA-1 is disallowed for Key Pair Generation and Digital Signature Generation, with the exception of digital signatures on ephemeral parameters in TLS.

Table 6 – Non-Approved Cryptographic Functions

Algorithm	Usage
RSA with key size smaller than 2048 bits	Key Pair Generation, Digital Signature Generation, Digital Signature Verification
Diffie-Hellman with key size smaller than 2048 bits	Key agreement
DSA with key size smaller than 2048 bits and greater than 3072 bits	Key Generation, Domain Parameters Generation and Verification, Digital Signature Generation and Verification
SM2	Digital Signature Generation and Verification
Chacha20 and Poly1305	Authenticated Data Encryption and Decryption
MD4	Message Digest
MD5	Message Digest
RMD160	Message Digest
HMAC with key size less than 112 bits	Message Authentication Code
ECDSA with non NIST curves.	Digital Signature Generation and Verification

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 can be used.
2. **Non-FIPS mode** (the non-Approved mode of operation): Approved and non-approved security functions can be used.

During installation the module is configured to work in FIPS mode. The module configurations are defined in section 9.1 Crypto Officer Guidance. The module enters FIPS mode of operation after power-up tests succeed.

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 API through which applications request services, and the TLS protocol internal state and messages sent and received from the TCP/IP protocol. The following table summarizes the four logical interfaces:

Table 7 – Ports and Interfaces

FIPS Interface	Physical Port	Logical Interface
Data input	Ethernet ports, keyboard	API input parameters, kernel I/O – network or files on file system, TLS protocol input messages
Data output	Ethernet ports, Display	API output parameters, kernel I/O – network or files on file system, TLS protocol output messages.
Control input	Keyboard, Serial port, Ethernet port, Network, keyboard	API function calls, API input parameters for control, TLS protocol internal state
Status output	Serial port, Ethernet port, Network, Display	API return codes, TLS protocol internal state.
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 8. The User and Crypto Officer roles are implicitly assumed by the entity accessing the module services.

Table 8 – Roles Description

Role ID	Role Description
Cryptographic Officer (CO)	performs module installation, initialization and certificates management.
User	performs cryptographic services (in both, FIPS and non-FIPS mode), TLS network protocol, key zeroization, show status and on-demand self-test.

3.2 Services

All services implemented by the module are listed in tables below. The approved services are shown in Table 9 and the non-approved services are shown in Table 10. Please note that the keys and Critical Security Parameters (CSPs) listed below 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.

Table 9– Approved Services

Service	Role	Description	Input	Output	CSP and Type of Access
Installation	CO	Module installation and configuration	None	None	None
Initialization	CO	Perform initialization of the module	API call parameters	Status	None
Zeroization	User	Zeroize and de-allocate memory containing sensitive data	Power cycle, Using provided APIs	Status	All CSP's – W
Show status	User	Return the current mode or state of the module	API call parameters	Status	None
Self-tests on-demand	User	Perform power-up self-tests	Power cycle	Status	None

Service	Role	Description	Input	Output	CSP and Type of Access
Certificates Management	CO	Generate certificates (Client, Server, CA)	API call parameters	Status, generated certificates	RSA public key – RX RSA private key – RX DSA public key – RX DSA private key – RX ECDSA public key – RX ECDSA private key – RX
Symmetric encryption and decryption	User	Encrypt plaintext using supplied key Decrypt ciphertext using supplied key	API call parameters, key, plaintext, ciphertext	Status, plaintext, ciphertext	AES key – RX AES GCM key – RX AES GCM IV – RX AES CCM – RX AES XTS – RX AES GCM – RX Triple-DES key – RX
RSA key generation	User	Generate and return a RSA key pair	API call parameters	Status, key pair	RSA public key – W RSA private key – W
RSA digital signature generation and verification	User	Generate a signature for the supplied message using the specified key. Verify the signature on the supplied message using the specified key.	API call parameters, key, signature, message	Status, signature, message	RSA public key – RX RSA private key – RX
DSA key generation	User	Generate and return a DSA key pair	API call parameters	Status, key pair	DSA public key – W DSA private key – W
DSA digital signature generation and verification	User	Generate a signature for the supplied message using the specified key. Verify the signature on the supplied message using the specified key.	API call parameters, key, signature, message	Status, signature, message	DSA public key – RX DSA private key – RX

Service	Role	Description	Input	Output	CSP and Type of Access
ECDSA key generation	User	Generate and return a ECDSA key pair	API call parameters	Status, key pair	ECDSA public key – W ECDSA private key – W
ECDSA signature generation and verification	User	Generate a signature for the supplied message using the specified key. Verify the signature on the supplied message using the specified key.	API call parameters, key, signature, message	Status, signature, message	ECDSA public key – RX ECDSA private key – RX
Random number generation	User	Generate random number	API call parameters	Status output, random value	DRBG Entropy Input – RX DRBG Seed – RX DRBG “C” Value – WRX DRBG “V” Value – WRX
Message digest	User	Compute and return a message digest using SHS algorithms	API call parameters, message	Status, hash	None
Message authentication code	User	Compute and return a message authentication code	API call parameters, key, message	Status, hash	HMAC key – RX AES CMAC key – RX AES GMAC key – RX Triple-DES CMAC key – RX
Key wrapping	User	AES Key wrapping (encryption and decryption)	API call parameters, key, plaintext key, ciphertext key	Status, plaintext key, ciphertext key	AES WP key – RX
Key encapsulation	User	RSA Key encapsulation (encryption and decryption)	API call parameters, key, plaintext key, ciphertext key	Status, plaintext key, ciphertext key	RSA public key – RX RSA private key – RX

Service	Role	Description	Input	Output	CSP and Type of Access
Diffie-Hellman Key Agreement	User	KAS FFC-SSC	API call parameters, key	Status, domain parameter, key	Domain parameters (p,q,g) – WRX DH public key – WRX DH private key – WRX
EC Diffie-Hellman Key Agreement	User	KAS ECC-SSC	API call parameters, key	Status, key components, key	ECDH public key – WRX ECDH private key – WRX
TLS network protocol v1.2	User	See Appendix I for the complete list of supported cipher suites.	API call parameters, internally derived	Status, Ciphered data	AES key – WRX AES GCM key – RX AES GCM IV – WRX RSA public key – RX ECDSA public key – RX DH public key – RX ECDH public key – RX RSA private key – RX ECDSA private key – RX DH private key – RX ECDH private key – RX Shared Secret - WRX

Table 10 – Non-Approved Services

Service	Role	Algorithms	CSP and Type of Access
Initialization	CO	None	None
Asymmetric encryption/decryption using keys disallowed by [SP800-131A]	User	RSA	RSA public key – RX RSA private key – RX DSA public key - RX DSA private key -RX
Symmetric encryption/decryption	User	DES-CBC mode DES-OFB mode DES-ECB mode Two key Triple-DES-CBC mode Two key Triple-DES-CFB mode	DES key - RX 2-key Triple-DES keys - RX

Service	Role	Algorithms	CSP and Type of Access
Key establishment using keys disallowed by [SP800-131A]	User	Diffie-Hellman, RSA	Domain parameters (p,q,g) – WRX DH public key – WRX DH private key – WRX RSA public key – RX RSA private key – RX
Message authentication code (MAC) using keys disallowed by [SP800-131A]	User	HMAC, CMAC	HMAC key - RX 2-key Triple DES key - RX
Message digest	User	MD4, MD5, RMD160	None
Digital signature generation and verification using keys or curves disallowed by [SP800-131A]	User	RSA, DSA, ECDSA	DSA public key – RX DSA private key – RX RSA public key – RX RSA private key – RX ECDSA public key – RX ECDSA private key – RX
Digital signature generation and verification	User	SM2	SM2 public and private keys – RX
Transport Layer Security network protocol v1.0, v1.1 and v1.3	User	Chacha20 and Poly1305, AES, SHS, MD5	AES – RX Chacha20 and Poly1305 - RX Shared Secret – WX

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 a commercially available general-purpose operating system executing on the hardware specified in Table 3 - Tested Platforms.

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 calling application's process space. The module only allows access to CSPs through its well-defined API.

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
AES Key	128, 192, 256-bit key	Input via API call parameter	Never exists the module	Not persistently stored	Unload module, Remove power or API call: EVP_CIPHER_CTX_free()	Encryption and decryption.
AES CMAC key	128, 192, 256-bit key	Input via API call parameter	Never exists the module	Not persistently stored	Unload module, Remove power or API call: EVP_CIPHER_CTX_free()	MAC generation and verification
AES CCM key	128, 192, 256-bit key	Input via API call parameter	Never exists the module	Not persistently stored	Unload module, Remove power or API call: EVP_CIPHER_CTX_free()	Encryption and decryption.
AES GCM/GMAC key	128, 192, 256-bit key	Input via API call parameter	Never exists the module	Not persistently stored	Unload module, Remove power or API call: EVP_CIPHER_CTX_free()	Encryption and decryption.
AES GCM IV	96-bits value	Generated internally, deterministically in compliance with TLS 1.2 GCM Cipher Suites for TLS and Section 8.2.1 of NIST SP 800-38D	Never exists the module	Not persistently stored	Unload module, Remove power or API call: EVP_CIPHER_CTX_free()	Initialization vector
AES XTS key	128 and 256-bits key	Input via API call parameter	Never exists the module	Not persistently stored	Unload module, Remove power or API call: EVP_CIPHER_CTX_free()	Encryption and decryption.
AES KW key	128, 192, 256-bit key	Input via API call parameter	Never exists the module	Not persistently stored	Unload module, Remove power or API call: EVP_CIPHER_CTX_free()	Encryption and decryption.
Triple-DES key	192-bit key	Input via API call parameter	Never exists the module	Not persistently stored	Unload module, Remove power or API call: EVP_CIPHER_CTX_free()	Encryption and decryption.
Triple-DES CMAC key	192-bit key	Input via API call parameter	Never exists the module	Not persistently stored	Unload module, Remove power or API call: EVP_CIPHER_CTX_free()	MAC generation and verification

CSP	CSP Type	Generation/ Input	Output	Storage	Zeroization	Use
HMAC keys	112-bit or greater	Input via API call parameter	Never exists the module	Not persistently stored	Unload module, Remove power or API call: HMAC_CTX_free()	Message authentication with SHA
RSA private key	2048, 3072, 4096-bit key	Internally generated, or input via API call parameter	Output in plaintext via API call parameter	Not persistently stored	Unload module, Remove power or API call: RSA_free()	Signature generation Decryption
DSA private key	224 and 256-bit key	Internally generated, or input via API call parameter	Output in plaintext via API call parameter	Not persistently stored	Unload module, Remove power or API call: DSA_free()	Signature generation
ECDSA private key	From 224-bit to 528 key sizes	Internally generated, or input via API call parameter	Output in plaintext via API call parameter	Not persistently stored	Unload module, Remove power or API call: EC_KEY_free()	Signature generation
Elliptic Curve Diffie-Hellman private keys	From 224-bit to 528 key sizes	Internally generated, or input via API call parameter	Output in plaintext via API call parameter	Not persistently stored	Unload module, Remove power or API call: EC_KEY_free()	Derive shared secret
Diffie-Hellman private Key	From 2048-bit to 8192 key sizes	Internally generated, or Input via API call parameter	Never exists to module	Not persistently stored	Unload module, Remove power or API call: DH_free()	TLS communication
Shared Secret	DH/ECDH shared secret	Internally derived	Never exits to module	Not persistently stored	Unload module, Remove power or API call: EVP_PKEY_CTX_free()	TLS communication
DRBG entropy	256 bits	Internally generated coming from entropy source	Never exits to module	Not persistently stored	Unload module, Remove power or API call: RAND_DRBG_free()	Entropy material for CTR_DRBG
DRBG seed	Random data – 384 bits	Internally generated using nonce along with DRBG entropy	Never exits to module	Not persistently stored	Unload module, Remove power or API call: RAND_DRBG_free()	Seeding material for DRBG
DRBG 'V' Value	Internal state value	Internally generated	Never exits the module	Not persistently stored	Unload module, Remove power or API call: RAND_DRBG_free()	Used for CTR_DRBG
DRBG key	Internal state value	Internally generated	Never exits to module	Not persistently stored	Unload module, Remove power or API call: RAND_DRBG_free()	Used for CTR_DRBG

Table 12 – Public Keys

Public Key	Type	Generation/ Input	Output	Storage	Zeroization	Use
RSA public key	2048, 3072, 4096-bit key	Internally generated or input via API call parameter	Output in plaintext via API call parameter	Not persistently stored	Unload module, Remove power or API call: RSA_free()	Signature verification Encryption
DSA public key	2048, 3072-bit key	Internally generated or input via API call parameter	Output in plaintext via API call parameter	Not persistently stored	Unload module, Remove power or API call: DSA_free()	Signature verification
ECDSA public key	P-224, P-256, P-384, P-521	Internally generated or input via API call parameter	Output in plaintext via API call parameter	Not persistently stored	Unload module, Remove power or API call: EC_KEY_free()	Signature verification
Elliptic Curve Diffie-Hellman public keys	From 224-bit to 528 key sizes	Internally generated or input via API call parameter	Output in plaintext via API call parameter	Not persistently stored	Unload module, Remove power or API call: EC_KEY_free()	Derive shared secret
Diffie-Hellman public Key	From 2048-bit to 8192 key sizes	Internally generated, or input via API call parameter	Output in plaintext via API call parameter	Not persistently stored	Unload module, Remove power or API call: DH_free()	TLS communication

The module provides AES-GCM services to a calling application in support of TLS 1.2 communications and supports acceptable GCM cipher suites from section 3.3.1 of NIST SP 800-52rev2. Available cipher suites are listed in section 11.

The module follows the TLS 1.2 protocol GCM IV generation method from section A.5 of the FIPS 140-2 Implementation Guidance. In compliance with RFC PKCS#1, the 96-bit AES-GCM IV consists of 32-bit name field and a 64-bit counter field. If the counter field exhausts the maximum number of possible values for a given key, then the calling application must trigger a new handshake to establish a new encryption key.

6.2 Random Number Generation

EulerOS-RNG consists of a NPTRNG and a DRNG in EulerOS 2.0. For the NPTRNG part, it is based on timing Jitter RNG, specifically Jitterentropy RNGD with version 3.0.1. The core structure in the NPTRNG is an entropy pool named jitter pool with size of 4096 bits. The purpose of the jitter pool is to store the random number collected by Jitterentropy RNGD. Meanwhile, this jitter pool is also used to output random numbers for a user-space interface (/dev/random).

The /dev/random from the Operational Environment is used as a source of random numbers for DRBG entropy input string. Specifically, the entropy is extracted from the /dev/random by using the getrandom() function of the Linux kernel (with GRND_RANDOM flag as parameter) to fill with entropy the [SP800-90A] compliant Deterministic Random Bit Generator (DRBG).

The module employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90A] for the creation of seeds for asymmetric keys and random numbers for the TLS protocol. In addition, the module provides a Random Number Generation service to calling applications.

The DRBG supports the CTR_DRBG mechanism. The DRBG is initialized during module initialization; the module loads by default the DRBG using the CTR_DRBG mechanism with AES-256 and derivation function with prediction resistance.

The module performs the DRBG health tests as defined in section 11.3 of [SP800-90A] at the module power up.

6.3 Key Generation

For generating RSA, DSA and ECDSA keys the module implements asymmetric key generation services compliant with [FIPS186-4]. A seed (i.e. the random value) used in asymmetric key generation is directly obtained from the [SP800-90A] DRBG according to [SP800-133].

The public and private key pairs used in the Diffie-Hellman and EC Diffie-Hellman SSC are generated internally by the module based on the requirements of the [SP800-56A].

6.4 Key Agreement / Key Transport / Key Derivation

The module provides Diffie-Hellman and EC Diffie-Hellman key agreement schemes. These key agreement schemes are also used as part of the TLS protocol key exchange. The module also provides key wrapping using the AES with KW mode and RSA key encapsulation using private key encryption and public key decryption primitives. RSA key encapsulation is also used as part of the TLS protocol key exchange.

The module provides approved key transport methods according to [IG] D.9. The key transport methods are provided either by:

- Using an approved key wrapping algorithm (e.g., AES-KW).
- An approved authenticated encryption mode (e.g., AES-GCM, AES-CCM).
- A combination method that occurs using an approved symmetric encryption mode (e.g., AES-128-CBC) together with an approved authentication method (HMAC).

According to Table 2: Comparable strengths in [SP 800-57], the key sizes of AES, RSA, Diffie-Hellman and EC Diffie-Hellman provides the following security strength in FIPS mode of operation:

- AES key wrapping provides between 128 and 256 bits of encryption strength.
- RSA key wrapping provides between 112 and 256 bits of encryption strength.
- Diffie-Hellman key agreement provides between 112 and 200 bits of encryption strength.
- EC Diffie-Hellman key agreement provides between 112 and 256 bits of encryption strength.

- Approved authenticated encryption mode key establishment methodology (AES-GCM, AES-CCM) provides between 128 and 256 bits of encryption strength.
- Combination of any approved AES encryption mode with and HMAC authentication key establishment methodology provides between 128 and 256 bits of encryption strength.
- Combination of approved Triple-DES encryption and HMAC authentication key establishment methodology provides 112 bits of encryption strength.

The module supports key derivation for the TLS protocol. The module implements the pseudo-random function (PRF) for TLSv1.2.

Note: As the module supports the size of RSA key pair and Diffie-Hellman domain parameters greater than 2048 bits up to 15360 bits or more, the encryption strength 256 bits is claimed for RSA key encapsulations and Diffie-Hellman key agreement.

6.5 Key Entry/Output

The keys are provided to the module via API input parameters in plaintext form and output via API output parameters in plaintext form.

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

6.6 Key/CSP Storage

Symmetric keys, HMAC keys, public and private keys are provided to the module by the calling application via API input parameters, and are destroyed by the module when invoking the appropriate API function calls.

As a software module, no physical storage is offered within the logical boundary, and therefore the module does not store any CSPs persistently beyond the lifetime of the API call. Any persistent key storage occurs outside the module's logical boundary but within the physical boundary and the management of these keys is responsibility of the calling application.

6.7 Key/CSP Zeroization

The memory occupied by keys is allocated by regular memory allocation operating system calls. The application is responsible for calling the appropriate zeroization functions provided in the module's API listed in Table 11.

The zeroization functions overwrite the memory occupied by keys with "zeros" and deallocate the memory with the regular memory deallocation operating system call. In case of abnormal termination, or swap in/out of a physical memory page of a process, the keys in physical memory are overwritten by the Linux kernel before the physical memory is allocated to another process.

7 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The Huawei EulerOS 2.0 OpenSSL 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

FIPS 140-2 requires that the module performs power-up tests to ensure the integrity of the module and the correctness of the cryptographic functionality at start up. In addition, some functions require continuous testing of the cryptographic functionality, such as the asymmetric key generation or the security functions critical.

See section 9.2.5 for descriptions of possible self-test errors and recovery procedures.

8.1 Power Up Self-tests

The module performs power-up tests when the module is loaded into memory, without operator intervention. Power-up tests ensure that the module is not corrupted and that the cryptographic algorithms work as expected.

The module has a default entry point (DEP) containing code that the OS loader executes automatically when the module is loaded into memory for execution before the main () function of the calling application.

While the module is executing the power-up tests, services are not available, and input and output are inhibited. The module is not available for use by the calling application until the power-up tests are completed successfully.

If any power-up test fails, the module returns the error code listed in Table 14 and displays the specific error message associated with the returned error code, and then enters in Critrial Error state. The subsequent calls to the module will also fail - thus no further cryptographic operations are possible. If the power-up tests complete successfully, the module will return 1 in the return code and the internal 'fips_selftest_fail' flag is set to 0.

8.1.1. Integrity Tests

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 critical error state.

8.1.2. Cryptographic Algorithm Tests

The module performs self-tests on all FIPS-Approved cryptographic algorithms supported in the Approved mode of operation, using the Known Answer Tests (KAT) and Pair-wise Consistency Tests (PCT) shown in the following table:

Table 13 – Power Up Self-tests

Algorithm	Description
AES	KATs: Encryption, Decryption Modes: ECB (key sizes: 128 bit), CCM (key size: 192 bits), GCM (key size: 256 bits), XTS (key sizes: 128 bits, 256 bits)

Algorithm	Description
	Key sizes: 128 bits, 192 bits, 256 bits
CMAC	KATs: Generation, Verification Key sizes: AES with 128, 192, 256 bits & Triple-DES with 3-key
DRBG	KAT: CTR_DRBG, with PR and DF Key size: AES with 256 bits
DSA	PCT: Signature Generation, Signature Verification Key size: 2048 bits
ECDSA	PCT: Signature Generation, Signature Verification Curve: P-256
HMAC	KATs: Message authentication SHS sizes: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512
RSA	KAT: Signature Generation, Signature Verification, RSA PKCS1-v1.5 KAT: Signature Generation, Signature Verification, RSA SHA256 PSS KAT: Public encrypt and private decrypt Key sizes: 2048 bits
SHS	KATs: SHA-1, SHA2-256, SHA2-512, SHA3-256, SHA3-512, SHAKE-128, SHAKE-256
TDES	KATs: Encryption, Decryption Modes: TECB, Key sizes: 3-keys
Diffie-Hellman	KATs: Primitive "Z" Computation
EC Diffie-Hellman	KATs: Primitive "Z" Computation

8.2 Pair-wise Consistency Tests

During the operating state of the module a Pair-wise Consistency Test (PCT) is performed just after the generation of a RSA, DSA or ECDSA key pair.

Table 14 – Pair-wise Consistency Tests

Algorithm	Description
DSA	PCT: Signature Generation, Signature Verification on every key pair generation
ECDSA	PCT: Signature Generation, Signature Verification on every key pair generation
RSA	PCT: Signature Generation, Signature Verification on every key pair generation PCT: Public encrypt and private decrypt on every key pair generation

8.3 Security Functions Critical Tests

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

- SP 800-90 DRBG Instantiate Test
- SP 800-90 DRBG Generate Test
- SP 800-90 DRBG Reseed Test

The module performs the conditional tests for assurances per [IG 9.6] for KAS-FFC and KAS-ECC.

8.4 On-Demand Self-tests

On-Demand self-tests can be invoked by powering-off and reloading the module which cause the module to run the power-up tests again. During the execution of the on-demand self-tests, services are not available and no data output or input is possible.

9 Guidance

9.1 Crypto Officer Guidance

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 in FIPS mode.

The operators can use the module in FIPS mode and Non-FIPS mode the instructions to activate these modes are described below:

- **Enable FIPS mode**
  1. Open the terminal and type:
     # fips-mode-setup --enable
  2. Reboot the operating system.
  3. Verify that FIPS Mode is enabled by running the command:
     # fips-mode-setup --check
     The response should be “FIPS mode is enabled.”

- **Enable Non-FIPS mode**
  1. Open the terminal and type:
     # fips-mode-setup --disable
  2. Reboot the operating system.
  3. Verify that FIPS Mode is disabled enabled by running the command:
     # fips-mode-setup --check
     The response should be “FIPS mode is disabled.”

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***

9.2 User Guidance

In order to run in FIPS mode, the module must be operated using the FIPS Approved services, with their corresponding FIPS Approved and FIPS allowed cryptographic algorithms provided in this Security Policy (see section 3.2 Services). In addition, key sizes must comply with [SP800-131A].

9.2.1 TLS

The module implements TLS versions 1.0, 1.1, 1.2 and 1.3. The use of TLS versions 1.0, 1.1 and 1.3 are not allowed in FIPS mode of operation.

The TLS protocol implementation provides both server and client sides. In order to operate in FIPS mode, digital certificates used for server and client authentication shall comply with the restrictions of

key size and message digest algorithms imposed by [SP800-131A]. In addition, as required also by [SP800-131A], Diffie-Hellman with keys smaller than 2048 bits must not be used.

The TLS server implementation allows the application to set the Diffie-Hellman key size. The server side must always set the DH parameters with the API call of SSL_CTX_set_tmp_dh(ctx, dh).

To comply with the FIPS 140-2 standard the requirement to not allow Diffie-Hellman key sizes smaller than 2048 bits must be met, to do this the Crypto Officer must ensure that:

1. In case the module is used as TLS server, the Diffie-Hellman parameters (dh argument) of the aforementioned API call must be 2048 bits or larger;
2. In case the module are used as TLS client, the TLS server must be configured to only offer Diffie-Hellman keys of 2048 bits or larger.

9.2.2 AES XTS

The only use of the AES algorithm in XTS mode is to store data on devices. As a restriction, the length of a single data unit encrypted with this algorithm shall not exceed 2^{20} AES blocks that is 16MB of data. Meanwhile, the module implements a check to ensure that the two AES keys used in this algorithm are not identical.

The AES algorithm in XTS mode can be only used for the cryptographic protection of data on storage devices, as specified in [SP800-38E]. The length of a single data unit encrypted with the XTS-AES shall not exceed 2^{20} AES blocks that is 16MB of data.

The module implements a check to ensure that the two AES keys used in XTS-AES algorithm are not identical.

9.2.3 Triple-DES

The user is responsible for ensuring that the same Triple-DES key shall not be used to encrypt more than 2^{16} 64-bit blocks of data.

9.2.4 AES-GCM IV

If the nonce_explicit part of the IV has been exhausted, the module will abort the TLS session and trigger a handshake to establish a new encryption key.

AES-GCM IV generation is compliant with RFC 5288. To ensure the module complies with IG A.5, AES-GCM can only be used in the context of TLSv1.2. Thus, the module is compliant with SP800-52.

In case the modules' power is lost and then restored, the key used for the AES GCM encryption/decryption shall be re-distributed.

9.2.5 Handling Self-Test Errors

When any of power-up self-tests or conditional self-test (except the conditional security functions critical) fail, the module will return an error code and enter in Critical Error state. The indicator for Critical Error is

a global error flag FIPS_selftest_fail. In Critical Error state, no cryptographic operation is available. To clear the error state, the CO must unload and reload the module (thus loading the module), and the power-up self-tests must then run to completion successfully.

The module reaches the Soft Error state when a conditional security function critical fails. The module indicates the error code and automatically clears this error state and transits to the normal calling application execution.

Table describes the error codes or messages and the events they are corresponding.

Table 15 – Error Codes and Events

Error code	Event
FIPS_R_FINGERPRINT_DOES_NOT_MATCH	The integrity test fails at the power-up phase.
FIPS_R_SELFTEST_FAILED	Any cryptographic operation is called when the module is in error state.
FIPS_R_PAIRWISE_TEST_FAILED	A PCT failed during key generation for DSA, ECDSA or RSA algorithms.
EC_F_EC_GROUP_CHECK	Ephemeral Unified key establishment scheme assurance test failure
DH_F_DH_CHECK_PARAMS_EX	dhEphem key establishment scheme assurance test failure
FIPS_F_FIPS_DRBG_INIT	The DRBG health tests has failed when instantiating the DRBG structure.
FIPS_F_FIPS_DRBG_CHEC-K	The DRBG health tests has failed when generating a new random data
FIPS_F_DRBG_RESEED	The DRBG health tests has failed when the reseed function has been invoked.

These errors are reported through the regular ERR interface of the shared libraries and can be queried by functions such as ERR_get_error().

As mentioned, once the module is in the Critical Error state, no further cryptographic operations are available, the module will return an error message: “FATAL FIPS SELFTEST FAILURE” printed to stderr and the module is terminated.

10 Mitigation of Other Attacks

10.1 Blinding Against RSA Timing Attacks

RSA is vulnerable to timing attacks. In a configuration where attackers can measure the time of RSA decryption or signature operations, blinding must be used to protect the RSA operation from that attack.

The module provides the API functions RSA_blinding_on() and RSA_blinding_off() to turn the blinding on and off for RSA. When the blinding is on, the module generates a random value to form a blinding factor in the RSA key before the RSA key is used in the RSA cryptographic operations.

Please note that the DRBG must be seeded prior to calling RSA_blinding_on() to prevent the RSA Timing Attack.

10.2 Weak Triple-DES Keys Detection

The module implements the DES_set_key_checked() for checking the weak Triple-DES key and the correctness of the parity bits when the Triple-DES key is going to be used in Triple-DES operations. The checking of the weak Triple-DES key is implemented in the API function DES_is_weak_key() and the checking of the parity bits is implemented in the API function DES_check_key_parity(). If the Triple-DES key does not pass the check, the module will return -1 to indicate the parity check error and -2 if the Triple-DES key matches to any value listed below:

```
static const DES_cblock weak_keys[NUM_WEAK_KEY]={
/* weak keys */
{0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01},
{0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE},
  {0x1F,0x1F,0x1F,0x1F,0x0E,0x0E,0x0E,0x0E},
{0xE0,0xE0,0xE0,0xE0,0xF1,0xF1,0xF1,0xF1},
  /* semi-weak keys */
  {0x01,0xFE,0x01,0xFE,0x01,0xFE,0x01,0xFE},
{0xFE,0x01,0xFE,0x01,0xFE,0x01,0xFE,0x01},
{0x1F,0xE0,0x1F,0xE0,0x0E,0xF1,0x0E,0xF1},
{0xE0,0x1F,0xE0,0x1F,0xF1,0x0E,0xF1,0x0E},
{0x01,0xE0,0x01,0xE0,0x01,0xF1,0x01,0xF1},
  {0xE0,0x01,0xE0,0x01,0xF1,0x01,0xF1,0x01},
  {0x1F,0xFE,0x1F,0xFE,0x0E,0xFE,0x0E,0xFE},
{0xFE,0x1F,0xFE,0x1F,0xFE,0x0E,0xFE,0x0E},
{0x01,0x1F,0x01,0x1F,0x01,0x0E,0x01,0x0E},
{0x1F,0x01,0x1F,0x01,0x0E,0x01,0x0E,0x01},
  {0xE0,0xFE,0xE0,0xFE,0xF1,0xFE,0xF1,0xFE},
{0xFE,0xE0,0xFE,0xE0,0xFE,0xF1,0xFE,0xF1}};
```

11 References and Definitions

Table 16 – 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)
[FIPS 202]	FIPS 202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions
[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 800-38B]	NIST SP 800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication
[SP 800-38C]	NIST SP 800-38C, Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality
[SP 800-38D]	NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
[SP 800-38E]	NIST SP 800-38E, Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices
[SP 800-38F]	NIST SP 800-38F, - Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping
[SP 800-52]	NIST SP 800-52, Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations
[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
[RFC 5288]	AES Galois Counter Mode (GCM) Cipher Suites for TLS

Table 17 – Acronyms

Acronym	Definition
AES	Advanced Encryption Standard
API	Application Program Interface
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CCM	Counter with Cipher Block Chaining-Message Authentication Code
CFB	Cipher Feedback
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DRGB	Deterministic Random Bit Generator
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
HMAC	Hash Message Authentication Code
KAS	Key Agreement Schema
KAT	Known Answer Test
KDF	Key Derivation Function
KW	Key Wrap
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
NDRNG	Non-Deterministic Random Number Generator
OFB	Output Feedback
PCT	Pair-wise Consistency Test
PRNG	Pseudo-Random Number Generator

Acronym	Definition
PSS	Probabilistic Signature Scheme
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
TLS	Transport Layer Security
XTS	XEX-based Tweaked-codebook mode with ciphertext Stealing

12 APPENDIX I

The module implements TLS versions 1.0, 1.1, 1.2 and 1.3. The use of TLS versions 1.0, 1.1 and 1.3 are not allowed in FIPS mode of operation, being only the use of version 1.2 approved in Approved mode of operation. Next the supported ciphersuites for each version are listed:

Supported ciphersuites for TLS v1.2:

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA

Supported ciphersuites for TLS v1.0, v1.1:

- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA

Supported ciphersuites for TLS v1.3:

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256