SafeLogic Inc.

CryptoComply for Java 140-3

FIPS 140-3 Non-Proprietary Security Policy

Software Version 2.0.0

Document Version 1.0

October 10, 2024

SafeLogic Inc.
530 Lytton Ave, Suite 200
Palo Alto, CA 94301
www.safelogic.com

Table of Contents

List of Tables

List of Figures

1 General Information

1.1 Overview

This document provides a non-proprietary FIPS 140-3 Security Policy for CryptoComply for Java 140-3.

SafeLogic Inc.'s CryptoComply for Java 140-3 is designed to provide FIPS 140-3 validated cryptographic functionality and is available for licensing. For more information, visit www.safelogic.com/cryptocomply.

1.1.1 About FIPS 140

Federal Information Processing Standards Publication 140-3, Security Requirements for Cryptographic Modules, (FIPS 140-3) specifies the latest requirements for cryptographic modules utilized to protect sensitive but unclassified information. The National Institute of Standards and Technology (NIST) and Canadian Centre for Cyber Security (CCCS) collaborate to run the Cryptographic Module Validation Program (CMVP), which assesses conformance to FIPS 140. NIST (through NVLAP) accredits independent testing labs to perform FIPS 140 testing. The CMVP reviews and validates modules tested against FIPS 140 criteria. *Validated* is the term given to a module that has successfully gone through this FIPS 140 validation process. Validated modules receive a validation certificate that is posted on the CMVP's website.

More information is available on the CMVP website at:
https://csrc.nist.gov/projects/cryptographic-module-validation-program.

1.1.2 About this Document

This non-proprietary cryptographic module Security Policy for CryptoComply for Java 140-3 from SafeLogic Inc. (SafeLogic) provides an overview of the product and a high-level description of how it meets the security requirements of FIPS 140-3. This document includes details on the module's cryptographic capabilities, services, sensitive security parameters, and self-tests. This Security Policy also includes guidance on operating the module while maintaining compliance with FIPS 140-3.

CryptoComply for Java 140-3 may also be referred to as "the module" in this document.

1.1.3 External Resources

The SafeLogic website (www.safelogic.com) contains information on SafeLogic services and products. The CMVP website maintains all FIPS 140 certificates for SafeLogic's FIPS 140 validations. These certificates also include SafeLogic contact information.

1.1.4 Notices

This document may be freely reproduced and distributed, but only in its entirety and without modification.

1.2 Security Levels

Table 1 lists the module's level of validation for each area in FIPS 140-3.

Table 1 - Security Levels

Section	Security Level
Overall Security Level	1
Section 1 – General Information	1
Section 2 – Cryptographic Module Specification	1
Section 3 – Cryptographic Module Interfaces	1
Section 4 – Roles, Services, and Authentication	1
Section 5 – Software/Firmware Security	1
Section 6 – Operational Environment	1
Section 7 – Physical Security	N/A
Section 8 – Non-Invasive Security	N/A
Section 9 – Sensitive Security Parameter Management	1
Section 10 – Self-Tests	1
Section 11 – Life-Cycle Assurance	1
Section 12 – Mitigation of Other Attacks	1

2 Cryptographic Module Specification

2.1 Description

Purpose and Use:

CryptoComply for Java 140-3 is a standards-based "Drop-in Compliance™" cryptographic module for Java environments.

The module delivers cryptographic services to host applications through a Java language Application Programming Interface (API).

Module Type: Software

Module Embodiment: Multi-Chip Stand Alone

Cryptographic Boundary:

The cryptographic boundary is the Java Archive (JAR) file, bc-fips-2.0.0.jar.

The module is the only component within the cryptographic boundary and the only component that carries out cryptographic functions covered by FIPS 140-3. The module classes are executed on the Java Virtual Machine (JVM) using the classes of the Java Runtime Environment (JRE). The JVM is the interface to the computer's Operating System (OS), which is the interface to the various physical components of the general purpose computer (GPC).

As a software cryptographic module, the module operates within the Tested Operational Environment's Physical Perimeter (TOEPP). The TOEPP physical perimeter is the physical perimeter of the GPC that the module operates on. The TOEPP includes the JVM/JRE, OS, and the GPC. The TOEPP includes the Operational Environment (OE) that the module operates in, the module itself, and all other applications that operate within the OE, including the host application for the module. The external entropy source used by the module is also within the TOEPP.

The module's block diagram is provided in Figure 1, which shows the cryptographic boundary and the logical relationship of the cryptographic module to the other software and hardware components of the TOEPP. The module's logical interfaces are defined by its API.

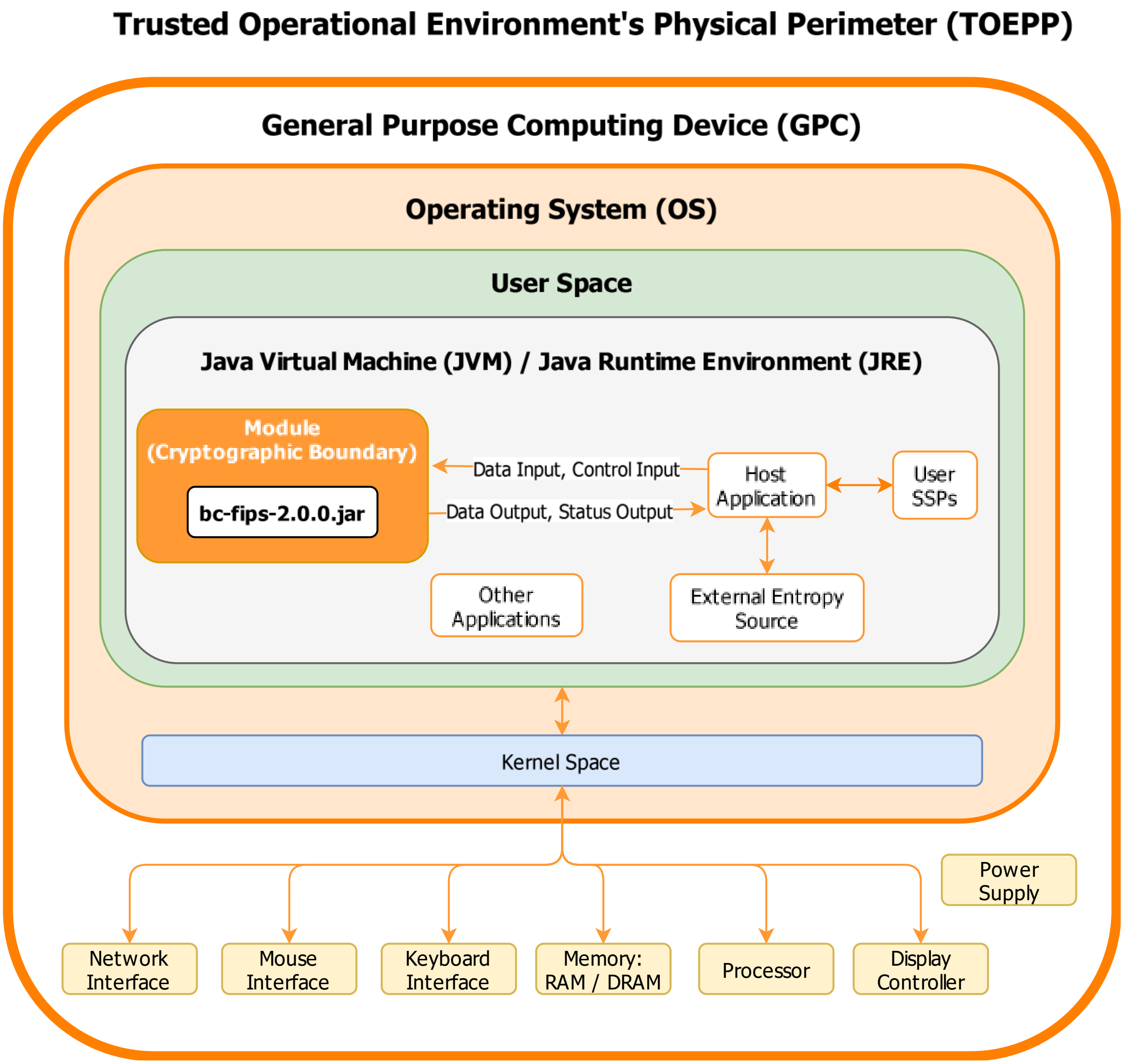

Figure 1 - Module Block Diagram

2.2 Tested and Vendor Affirmed Module Version and Identification

Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets):

Table 2 - Executable Code Sets

Package/File Names	Software/ Firmware Version	Integrity Test Implemented
bc-fips-2.0.0.jar	2.0.0	HMAC-SHA-256

Confirming the Module Checksum, Functionality, and Versioning

The module checksum, functionality, and versioning can be confirmed by executing the command:

java -cp bc-fips-2.0.0.jar org.bouncycastle.util.DumpInfo

which should display:

Version Info: BouncyCastle Security Provider (FIPS edition) v2.0.0

FIPS Ready Status: READY

Module SHA-256 HMAC:
164c8ae41945cb85fdc65666fc4de7301a65d29659ecd455ee5199c7d42d107e

This display indicates that the JAR represents the software release bc-fips-2.0.0, that it has successfully passed all its startup tests, and that the software release is confirmed to have the HMAC listed above.

Tested Operational Environments - Software, Firmware, Hybrid:

The module operates in a modifiable operational environment under the FIPS 140-3 definitions. The cryptographic module was tested on the following operational environments on the GPC platforms detailed in Table 3.

Table 3 - Tested Operational Environments – Software/Firmware/Hybrid

Operating System	Hardware Platform	Processor(s)	PAA/PAI	Version(s)
VMware Photon OS 4.0 with JRE 8 on VMware ESXi 8.0	Dell PowerEdge R650	Intel Xeon Gold 6330	No	2.0.0
VMware Photon OS 4.0 with JRE 11 on VMware ESXi 8.0	Dell PowerEdge R650	Intel Xeon Gold 6330	No	2.0.0
VMware Photon OS 4.0 with JRE 17 on VMware ESXi 8.0	Dell PowerEdge R650	Intel Xeon Gold 6330	No	2.0.0
VMware Photon OS 5.0 with JRE 21 on VMware ESXi 8.0	Dell PowerEdge R650	Intel Xeon Gold 6330	No	2.0.0

Vendor-Affirmed Operational Environments - Software, Firmware, Hybrid:

Porting guidance is defined in the FIPS 140-3 CMVP Management Manual Section 7.9. The cryptographic module will remain compliant with the FIPS 140-3 validation when operating on any GPC provided that:

- No source code modifications were made
- The module operates on any general-purpose platform/processor that supports the specified operating system as listed on the validation entry. Or the module uses another compatible platform, such as one of the Java SE Runtime Environments listed in the table below (Table 4).

The CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when so ported if the specific operational environment is not listed on the validation certificate.

Table 4 - Vendor Affirmed Operational Environments – Software/Firmware/Hybrid

#	Operating System	Hardware Platform
1.	Java SE Runtime Environment v8 (1.8) with HP-UX	Generic Hardware Platform
2.	Java SE Runtime Environment v11 (1.11) with HP-UX	Generic Hardware Platform
3.	Java SE Runtime Environment v17 (1.17) with HP-UX	Generic Hardware Platform
4.	Java SE Runtime Environment v21 (21) with HP-UX	Generic Hardware Platform
5.	Java SE Runtime Environment v8 (1.8) with Linux CentOS	Generic Hardware Platform
6.	Java SE Runtime Environment v11 (1.11) with Linux CentOS	Generic Hardware Platform
7.	Java SE Runtime Environment v17 (1.17) with Linux CentOS	Generic Hardware Platform
8.	Java SE Runtime Environment v21 (21) with Linux CentOS	Generic Hardware Platform
9.	Java SE Runtime Environment v8 (1.8) with Red Hat Enterprise Linux	Generic Hardware Platform
10.	Java SE Runtime Environment v11 (1.11) with Red Hat Enterprise Linux	Generic Hardware Platform
11.	Java SE Runtime Environment v17 (1.17) with Red Hat Enterprise Linux	Generic Hardware Platform
12.	Java SE Runtime Environment v21 (21) with Red Hat Enterprise Linux	Generic Hardware Platform
13.	Java SE Runtime Environment v8 (1.8) with Linux Debian	Generic Hardware Platform
14.	Java SE Runtime Environment v11 (1.11) with Linux Debian	Generic Hardware Platform
15.	Java SE Runtime Environment v17 (1.17) with Linux Debian	Generic Hardware Platform
16.	Java SE Runtime Environment v21 (21) with Linux Debian	Generic Hardware Platform
17.	Java SE Runtime Environment v8 (1.8) with Linux Fedora	Generic Hardware Platform
18.	Java SE Runtime Environment v11 (1.11) with Linux Fedora	Generic Hardware Platform
19.	Java SE Runtime Environment v17 (1.17) with Linux Fedora	Generic Hardware Platform
20.	Java SE Runtime Environment v21 (21) with Linux Fedora	Generic Hardware Platform
21.	Java SE Runtime Environment v8 (1.8) with Linux Oracle RHC	Generic Hardware Platform
22.	Java SE Runtime Environment v11 (1.11) with Linux Oracle RHC	Generic Hardware Platform
23.	Java SE Runtime Environment v17 (1.17) with Linux Oracle RHC	Generic Hardware Platform
24.	Java SE Runtime Environment v21 (21) with Linux Oracle RHC	Generic Hardware Platform
25.	Java SE Runtime Environment v8 (1.8) with Linux Oracle UEK	Generic Hardware Platform
26.	Java SE Runtime Environment v11 (1.11) with Linux Oracle UEK	Generic Hardware Platform
27.	Java SE Runtime Environment v17 (1.17) with Linux Oracle UEK	Generic Hardware Platform
28.	Java SE Runtime Environment v21 (21) with Linux Oracle UEK	Generic Hardware Platform
29.	Java SE Runtime Environment v17 (1.8) with Linux Photon	Generic Hardware Platform
30.	Java SE Runtime Environment v11 (1.11) with Linux Photon	Generic Hardware Platform
31.	Java SE Runtime Environment v17 (1.17) with Linux Photon	Generic Hardware Platform
32.	Java SE Runtime Environment v21 (21) with Linux Photon	Generic Hardware Platform
33.	Java SE Runtime Environment v8 (1.8) with Linux SUSE	Generic Hardware Platform
34.	Java SE Runtime Environment v11 (1.11) with Linux SUSE	Generic Hardware Platform
35.	Java SE Runtime Environment v17 (1.17) with Linux SUSE	Generic Hardware Platform
36.	Java SE Runtime Environment v21 (21) with Linux SUSE	Generic Hardware Platform

#	Operating System	Hardware Platform
37.	Java SE Runtime Environment v8 (1.8) with Linux Ubuntu	Generic Hardware Platform
38.	Java SE Runtime Environment v11 (1.11) with Linux Ubuntu	Generic Hardware Platform
39.	Java SE Runtime Environment v17 (1.17) with Linux Ubuntu	Generic Hardware Platform
40.	Java SE Runtime Environment v21 (21) with Linux Ubuntu	Generic Hardware Platform
41.	Java SE Runtime Environment v8 (1.8) with Mac OS X	Generic Hardware Platform
42.	Java SE Runtime Environment v11 (1.11) with Mac OS X	Generic Hardware Platform
43.	Java SE Runtime Environment v8 (1.8) with Microsoft Windows	Generic Hardware Platform
44.	Java SE Runtime Environment v11 (1.11) with Microsoft Windows	Generic Hardware Platform
45.	Java SE Runtime Environment v17 (1.17) with Microsoft Windows	Generic Hardware Platform
46.	Java SE Runtime Environment v21 (21) with Microsoft Windows	Generic Hardware Platform
47.	Java SE Runtime Environment v8 (1.8) with Microsoft Windows Server	Generic Hardware Platform
48.	Java SE Runtime Environment v11 (1.11) with Microsoft Windows Server	Generic Hardware Platform
49.	Java SE Runtime Environment v17 (1.17) with Microsoft Windows Server	Generic Hardware Platform
50.	Java SE Runtime Environment v21 (21) with Microsoft Windows Server	Generic Hardware Platform
51.	Java SE Runtime Environment v8 (1.8) with Microsoft Windows XP	Generic Hardware Platform
52.	Java SE Runtime Environment v11 (1.11) with Microsoft Windows XP	Generic Hardware Platform
53.	Java SE Runtime Environment v17 (1.17) with Microsoft Windows XP	Generic Hardware Platform
54.	Java SE Runtime Environment v21 (21) with Microsoft Windows XP	Generic Hardware Platform
55.	Java SE Runtime Environment v8 (1.8) with Solaris	Generic Hardware Platform
56.	Java SE Runtime Environment v11 (1.11) with Solaris	Generic Hardware Platform
57.	Java SE Runtime Environment v17 (1.17) with Solaris	Generic Hardware Platform
58.	Java SE Runtime Environment v21 (21) with Solaris	Generic Hardware Platform
59.	Java SE Runtime Environment v8 (1.8) with AIX	Generic Hardware Platform
60.	Java SE Runtime Environment v11 (1.11) with AIX	Generic Hardware Platform
61.	Java SE Runtime Environment v17 (1.17) with AIX	Generic Hardware Platform
62.	Java SE Runtime Environment v21 (21) with AIX	Generic Hardware Platform
63.	Java SE Runtime Environment v17 (1.17) with Red Hat Enterprise Linux	Generic Hardware Platform with Intel Cascade Lakes
64.	Java SE Runtime Environment v21 (21) with Red Hat Enterprise Linux	Generic Hardware Platform with Intel Cascade Lakes
65.	Java SE Runtime Environment v17 (1.17) with Red Hat Enterprise Linux	Generic Hardware Platform with Intel Sapphire Rapids
66.	Java SE Runtime Environment v21 (21) with Red Hat Enterprise Linux	Generic Hardware Platform with Intel Sapphire Rapids
67.	Java SE Runtime Environment v17 (1.17) with Ubuntu	Generic Hardware Platform with Intel Cascade Lakes
68.	Java SE Runtime Environment v21 (21) with Ubuntu	Generic Hardware Platform with Intel Cascade Lakes

#	Operating System	Hardware Platform
69.	Java SE Runtime Environment v17 (1.17) with Ubuntu	Generic Hardware Platform with Intel Sapphire Rapids
70.	Java SE Runtime Environment v21 (21) with Ubuntu	Generic Hardware Platform with Intel Sapphire Rapids
71.	Java SE Runtime Environment v17 (1.17) with ClevOS	Generic Hardware Platform with Intel Cascade Lake
72.	Java SE Runtime Environment v21 (21) with ClevOS	Generic Hardware Platform with Intel Cascade Lakes
73.	Java SE Runtime Environment v17 (1.17) with ClevOS	Generic Hardware Platform with Intel Sapphire Rapids
74.	Java SE Runtime Environment v21 (21) with ClevOS	Generic Hardware Platform with Intel Sapphire Rapids
75.	Java SE Runtime Environment v17 (1.17) with ClevOS	Generic Hardware Platform with Intel Haswell
76.	Java SE Runtime Environment v21 (21) with ClevOS	Generic Hardware Platform with Intel Haswell
77.	Java SE Runtime Environment v17 (1.17) with ClevOS	Generic Hardware Platform with Intel Broadwell
78.	Java SE Runtime Environment v21 (21) with ClevOS	Generic Hardware Platform with Intel Broadwell

2.3 Excluded Components

Not applicable.

2.4 Modes of Operation

Modes List and Description:

Table 5 - Modes of Operation

Name	Description	Type	Status Indicator
Approved mode	Only supports approved operations	Approved	*CryptoServicesRegistrar.IsInApprovedOnlyMode()* can be called to determine the mode of operation. This method will return true for approved mode.
Non-approved mode	Permits operations that are not approved	Non-Approved	*CryptoServicesRegistrar.IsInApprovedOnlyMode()* can be called to determine the mode of operation. This method will return false for non-approved mode.

Mode Change Instructions and Status:

In default operation the module will start with all algorithms and services enabled.

If the module detects that the system property *org.bouncycastle.fips.approved_only* is set to *true* the module will start in approved mode and non-approved mode functionality will not be available.

The module optionally uses the Java SecurityManager. If the underlying JVM is running with a Java SecurityManager installed the module starts in approved mode by default with secret and private key export disabled. When the module is not used within the context of the Java SecurityManager, it will start by default in the non-approved mode. Refer to Security Policy Section 11.3 for additional information about the Java SecurityManager.

Refer to Security Policy Section 11.4.1 for additional information on the module's mode of operation rules.

2.5 Algorithms

The module implements the algorithms specified in the tables below. The module supports both an Approved mode and a Non-approved mode of operation. Please see Security Policy Section 2.4 for additional details on the modes of operation and the configuration of the Approved mode of operation. Please see Security Policy Section 11.1 for Initialization steps.

2.5.1 Approved Algorithms

The module implements the following approved algorithms that have been tested by the Cryptographic Algorithm Validation Program (CAVP). 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 6 - Approved Algorithms, CAVP Tested

Algorithm Name (Implementation)	CAVP Cert Name	Algorithm Properties	Reference	Use/Function
AES	A4399	Modes: CBC, CFB8, CFB128, CTR, ECB, FF1, OFB Key sizes: 128, 192, 256 bits	AES [FIPS 197, SP 800-38A], AES FF1 Format Preserving Encryption [SP 800-38G]	Encryption, Decryption
AES CBC Ciphertext Stealing (CS)	A4399	Modes: CBC-CS1, CBC-CS2, CBC-CS3 Key sizes: 128, 192, 256 bits	[Addendum to SP 800-38A, Oct 2010]	Encryption, Decryption
AES CCM	A4399	Key sizes: 128, 192, 256 bits	[SP 800-38C]	Generation, Authentication
AES CMAC	A4399	Key sizes: 128, 192, 256 bits	[SP 800-38B]	Generation, Authentication

Algorithm Name (Implementation)	CAVP Cert Name	Algorithm Properties	Reference	Use/Function
AES GCM/GMAC[1]	A4399	Key sizes: 128, 192, 256 bits	[SP 800-38D]	Generation, Authentication
AES KW, KWP (KTS: Key Wrapping Using AES[2])	A4399	Modes: AES KW, KWP Key sizes: 128, 192, 256 bits (key establishment methodology providing 128, 192 or 256 bits of encryption strength)	[SP 800-38F]	Key Wrapping
DRBG, Counter DRBG	A4399	AES 128, AES 192, AES 256	[SP 800-90Ar1]	Random Bit Generation
DRBG, Hash DRBG	A4399	SHA sizes: SHA-1, SHA-224, SHA-256, SHA-384, SHA2-512, SHA-512/224, SHA2-512/256	[SP 800-90Ar1]	Random Bit Generation
DRBG, HMAC DRBG	A4399	SHA sizes: SHA-1, SHA-224, SHA-256, SHA-384, SHA2-512, SHA-512/224, SHA2-512/256	[SP 800-90Ar1]	Random Bit Generation
DSA[3]	A4399	Key sizes: 1024[4], 2048, 3072 bits	[FIPS 186-4]	Key Pair Generation, PQG Generation, PQG Verification, Signature Generation, Signature Verification
ECDSA	A4399	Curves/Key sizes: P-192, P-224, P-256, P-384, P-521, K-163, K-233, K-283, K-409, K-571, B-163, B-233, B-283, B-409, B-571[5]	[FIPS 186-4]	Key Generation, Key Verification, Signature Generation, Signature Verification

[1] GCM encryption with an internally generated IV, see Security Policy Section 2.6.1 concerning external IVs. IV generation is compliant with IG C.H.
[2] Keys are not established directly into the module using key agreement or key transport algorithms.
[3] DSA signature generation with SHA-1 is only for use with protocols.
[4] Key size only used for Signature Verification
[5] Curves P-192, K-163, and B-163 only used for Signature Verification and Key Verification

Algorithm Name (Implementation)	CAVP Cert Name	Algorithm Properties	Reference	Use/Function
HMAC	A4399	SHA sizes: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	[FIPS 198-1]	Generation, Authentication
KAS-ECC[6]	A4399	Domain Parameter Generation Methods/Schemes: P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571 ephemeralUnified, fullMqv, fullUnified, onePassDh, onePassMqv, onePassUnified, staticUnified Curves specified above providing between 112 and 256 bits of encryption strength	[SP 800-56Ar3]	Key Agreement
KAS-FFC[6]	A4399	Domain Parameter Generation Methods/Schemes: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 dhHybrid1, MQV2, dhEphem, dhHybrid, OneFlow, MQV1, dhOneFlow, dhStatic Groups specified above providing between 112 and 200 bits of encryption strength	[SP 800-56Ar3]	Key Agreement
KAS-IFC	A4399	RSASVE with, and without, key confirmation. Key sizes: 2048, 3072, 4096 providing between 112 and 152 bits of encryption strength	[SP 800-56Br2, Section 7.2.1]	Key Agreement

[6] Keys are not established directly into the module using key agreement or key transport algorithms.

Algorithm Name (Implementation)	CAVP Cert Name	Algorithm Properties	Reference	Use/Function
KDA, HKDF	A4399	PRFs: HMAC-SHA-1, HMAC SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, HMAC-SHA-512/224, HMAC-SHA-512/256, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512	[SP 800-56Cr2]	Key Derivation
KDA, One Step	A4399	PRFs: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512, HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, HMAC-SHA-512/224, HMAC-SHA-512/256, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512, KMAC-128, KMAC-256	[SP 800-56Cr2]	Key Derivation
KDA, Two Step	A4399	PRFs: HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, HMAC-SHA-512/224, HMAC-SHA-512/256, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512, CMAC-AES128, CMAC-AES192, CMAC-AES256	[SP 800-56Cr2]	Key Derivation
KDF, using Pseudorandom Functions[7]	A4399	Modes: Counter Mode, Feedback Mode, Double-Pipeline Iteration Mode Types: CMAC-based KBKDF with AES (128, 192, 256) HMAC-based KBKDF with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	[SP 800-108]	Key Derivation

[7] Note: CAVP testing is not provided for use of the PRFs SHA-512/224 and SHA-512/256. These must not be used in approved mode.

Algorithm Name (Implementation)	CAVP Cert Name	Algorithm Properties	Reference	Use/Function
KDF, Existing Application-Specific[8]	CVL A4399	TLS v1.0/1.1 KDF SHA sizes: SHA2-256, SHA2-384, SHA2-512	[SP 800-135r1]	Key Derivation
KDF, Existing Application-Specific[8]	CVL A4399	TLS 1.2 KDF SHA sizes: SHA2-256, SHA2-384, SHA2-512	[SP 800-135r1]	Key Derivation
KDF, Existing Application-Specific[8]	CVL A4399	SNMP KDF Password Length: 64, 8192	[SP 800-135r1]	Key Derivation
KDF, Existing Application-Specific[8]	CVL A4399	SSH KDF AES: 128 SHA sizes: SHA2-224	[SP 800-135r1]	Key Derivation
KDF, Existing Application-Specific[8]	CVL A4399	X9.63 KDF SHA sizes: SHA2-224, SHA2-256, SHA2-384, SHA2-512	[SP 800-135r1]	Key Derivation Can be used along with KAS-SSC
KDF, Existing Application-Specific[8]	CVL A4399	IKEv2 KDF SHA sizes: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	[SP 800-135r1]	Key Derivation
KDF, Existing Application-Specific[8]	CVL A4399	SRTP KDF AES: 128, 192, 256	[SP 800-135r1]	Key Derivation
KTS-IFC	A4399	RSA-OAEP with, and without, key confirmation. Key sizes: 2048, 3072, 4096 providing between 112 and 152 bits of encryption strength Key Generation Method: rsakpg2-crt	[SP 800-56Br2, Section 7.2.2]	Key Transport
PBKDF, Password-based	A4399	Options: PBKDF with Option 1a Types: HMAC-based KDF using SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	[SP 800-132]	Key Derivation
RSA	A4399	Key sizes: 2048, 3072, 4096	[FIPS 186-4, ANSI X9.31-1998 and PKCS #1 v2.1 (PSS and PKCS1.5)]	Key Pair Generation

[8] No parts of the protocols (TLS, SNMPv3, SSHv2, X9.63, IKEv2, SRTP), other than the approved cryptographic algorithms and the KDFs, have been reviewed or tested by the CAVP and CMVP

Algorithm Name (Implementation)	CAVP Cert Name	Algorithm Properties	Reference	Use/Function
RSA	A4399	Key sizes: 2048, 3072, 4096	[FIPS 186-4, ANSI X9.31-1998 and PKCS #1 v2.1 (PSS and PKCS1.5)]	Signature Generation
RSA	A4399	Key sizes: 1024, 2048, 3072, 4096	[FIPS 186-4, ANSI X9.31-1998 and PKCS #1 v2.1 (PSS and PKCS1.5)]	Signature Verification
RSA	A4399	Key sizes: 1024, 1536, 2048, 3072, 4096	[FIPS 186-2, ANSI X9.31-1998 and PKCS #1 v2.1 (PSS and PKCS1.5)]	Signature Verification
RSA Decryption Primitive	CVL A4399	Key size: 2048	[SP 800-56Br2]	Component Test
RSA Signature Primitive	CVL A4399	Key size: 2048	[FIPS 186-4]	Component Test
Safe Primes	A4399	Parameter sets: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	[SP 800-56Ar3]	Key Generation, Key Verification
SHA-3, SHAKE	A4399	SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256	[FIPS 202]	Digital Signature Generation, Digital Signature Verification, non-Digital Signature Applications

Algorithm Name (Implementation)	CAVP Cert Name	Algorithm Properties	Reference	Use/Function
SHA-3 Derived Functions	A4399	Types: cSHAKE-128, cSHAKE-256, KMAC-128, KMAC-256, ParallelHash-128, ParallelHash-256, TupleHash-128, TupleHash-256	[SP 800-185]	Digital Signature Generation, Digital Signature Verification, non-Digital Signature Applications
SHS	A4399	SHA sizes: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256	[FIPS 180-4]	Digital Signature Generation, Digital Signature Verification, non-Digital Signature Applications

2.5.2 Vendor Affirmed Algorithms

Vendor-Affirmed Algorithms:

Table 7 - Vendor Affirmed Algorithms

Algorithm Name	Algorithm Properties	Implementation	Reference
CKG	Used for the generation of symmetric keys and asymmetric seeds	Other Cryptographic key generation	[SP 800-133r2] CKG using output from DRBG, Vendor Affirmed per IG D.H. The resulting key or a generated seed is an unmodified output from a DRBG: • Section 5.1 (Asymmetric seeds from DRBG) • Section 6.1 (Direct Generation of Symmetric keys from DRBG)

2.5.3 Non-Approved, Allowed Algorithms

Non-Approved, Allowed Algorithms:

Not applicable.

2.5.4 Non-Approved, Allowed Algorithms with No Security Claimed

Non-Approved, Allowed Algorithms with No Security Claimed.

These algorithms are Allowed in Approved mode.

Table 8 - Non-Approved, Allowed Algorithms with No Security Claimed

Algorithm	Caveat	Use/Function
MD5 within TLS	Allowed per IG 2.4.A, no security claimed	MD5 used within a TLS handshake

2.5.5 Non-Approved, Not Allowed Algorithms

Non-Approved, Not Allowed Algorithms:

Table 9 - Non-Approved, Not Allowed Algorithms

Algorithm	Use/Function
AES (non-compliant[9])	Non-approved modes for AES
ARC4 (RC4)	ARC4/RC4 stream cipher
Blowfish	Blowfish block cipher
Camellia	Camellia block cipher
CAST5	CAST5 block cipher
ChaCha20	ChaCha20 stream cipher
ChaCha20-Poly1305	AEAD ChaCha20 using Poly1305 as the MAC
DES	DES block cipher
Diffie-Hellman KAS (non-compliant[10])	non-compliant key agreement methods
DSA (non-compliant[11])	non-FIPS digest signatures using DSA
DSTU4145	DSTU4145 EC algorithm
ECDSA (non-compliant[12])	non-FIPS digest signatures using ECDSA

[9] Support for additional modes of operation.
[10] Support for additional key sizes and the establishment of keys of less than 112 bits of security strength.
[11] Deterministic signature calculation, support for additional digests, and key sizes.
[12] Deterministic signature calculation, support for additional digests, and key sizes.

Algorithm	Use/Function
EdDSA	Ed25519 and Ed448 signature algorithms
ElGamal	ElGamal key transport algorithm
FF3-1	Format Preserving Encryption – AES FF3-1
GOST28147	GOST-28147 block cipher
GOST3410-1994	GOST-3410-1994 algorithm
GOST3410-2001	GOST-3410-2001 EC algorithm
GOST3410-2012	GOST-3410-2012 EC algorithm
GOST3411	GOST-3411-1994 message digest
GOST3411-2012-256	GOST-3411-2012 256-bit message digest
GOST3411-2012-512	GOST-3411-2012 512-bit message digest
HMAC-GOST3411	GOST-3411 HMAC
HMAC-MD5	MD5 HMAC
HMAC-RIPEMD128	RIPEMD128 HMAC
HMAC-RIPEMD160	RIPEMD160 HMAC
HMAC-RIPEMD256	RIPEMD256HMAC
HMAC-RIPEMD320	RIPEMD320 HMAC
HMAC-TIGER	TIGER HMAC
HMAC-WHIRLPOOL	WHIRLPOOL HMAC
HSS	HSS signature scheme (RFC 8708)
IDEA	IDEA block cipher
KAS[13] using SHA-512/224 or SHA-512/256 (non-compliant)	Key Agreement using SHA-512/224 and SHA-512/256 based KDFs
KBKDF using SHA-512/224 or SHA-512/256 (non-compliant)	KBKDF2 using the PRFs SHA-512/224 and SHA-512/256
LMS	LMS signature scheme (RFC 8708)
MD5	MD5 message digest
OpenSSL PBKDF (non-compliant)	OpenSSL PBE key derivation scheme
PKCS#12 PBKDF (non-compliant)	PKCS#12 PBE key derivation scheme
PKCS#5 Scheme 1 PBKDF (non-compliant)	PKCS#5 PBE key derivation scheme
Poly1305	Poly1305 message MAC

[13] Keys are not directly established into the module using key agreement or transport techniques.

Algorithm	Use/Function
PRNG X9.31	X9.31 PRNG
RC2	RC2 block cipher
RIPEMD128	RIPEMD128 message digest
RIPEMD160	RIPEMD160 message digest
RIPEMD256	RIPEMD256 message digest
RIPEMD320	RIPEMD320 message digest
RSA (non-compliant[14])	Non-compliant RSA signature schemes
RSA KTS (non-compliant[15])	Non-compliant RSA key transport schemes
SCrypt (non-compliant)	SCrypt using non-compliant PBKDF2
SEED	SEED block cipher
Serpent	Serpent block cipher
SipHash	SipHash MAC
SHACAL-2	SHACAL2 block cipher
TIGER	TIGER message digest
Triple-DES	Triple-DES cipher
Twofish	Twofish block cipher
WHIRLPOOL	WHIRLPOOL message digest
XDH	X25519 and X448 key agreement algorithms

2.6 Algorithm Specific Information

2.6.1 Enforcement and Guidance for GCM IVs (IG C.H conformance)

IVs for GCM can be generated randomly, or via a FipsNonceGenerator. IV generation is compliant with IG C.H.

Where an IV is not generated within the module the module supports the importing of GCM IVs. In approved mode, importing a GCM IV for encryption that originates from outside the module is non-conformant.

In approved mode, when a GCM IV is generated randomly, the module enforces the use of an approved DRBG in line with Section 8.2.2 of SP 800-38D.

[14] Support for additional digests and signature formats, PKCS#1 1.5 key wrapping, support for additional key sizes.
[15] Support for additional key sizes and the establishment of keys of less than 112 bits of security strength.

In approved mode, when a GCM IV is generated using the FipsNonceGenerator, a counter is used as the basis for the nonce and the IV is generated in accordance with TLS protocol. Rollover of the counter in the FipsNonceGenerator will result in an IllegalStateException indicating the FipsNonceGenerator is exhausted and (as per IG C.H) where used for TLS 1.2, rollover will terminate any TLS session in process using the current key and the exception can only be recovered from by using a new handshake and creating a new FipsNonceGenerator.

A service indicator for IV usage is provided in the module through Java logging. Setting the logging level to Level.FINE for the named logger *org.bouncycastle.jcajce.provider.BaseCipher* will produce a log message when an IV which may have been produced outside the module and/or not from a compliant source is detected. The log message will be of the standard form including the detail:

FINE: Passed in GCM nonce detected: <IV value>

where <IV value> is a HEX representation of the IV in use.

Setting the logging level to Level.FINER will produce an additional log message for any GCM IV which is used if the previous Level.FINE message is not activated. Log messages in this case will show the detail as:

FINER: GCM nonce detected: <IV value>

where <IV value> is a HEX representation of the IV in use.

Per IG C.H, this Security Policy also states that in the event module power is lost and restored the consuming application must ensure that any of its AES GCM keys used for encryption or decryption are re-distributed.

The AES GCM mode falls under:

- IG C.H scenario 2: GCM IV is generated randomly, and the module uses an Approved DRBG that is internal to the module’s boundary. The IV length is 96 bits.
- IG C.H scenario 1 for TLS v1.2 protocol: The module is compatible with the TLS v1.2 protocol and supports acceptable AES GCM ciphersuites from Section 3.3.1 of the SP 800-52r2.

2.6.2 Enforcement and Guidance for Use of the Approved PBKDF (IG D.N conformance)

The PBKDF aligns with Option 1a in Section 5.4 of SP 800-132.

In line with the requirements for SP 800-132, keys generated using the approved PBKDF must only be used for storage applications. Any other use of the approved PBKDF is non-conformant.

In approved mode the module enforces that any password used must encode to at least 14 bytes (112 bits) and that the salt is at least 16 bytes (128 bits) long. The iteration count associated with the PBKDF should be as large as practical.

As the module is a general purpose software module, it is not possible to anticipate all the levels of use for the PBKDF, however a user of the module should also note that a password should at least contain enough entropy to be unguessable and also contain enough entropy to reflect the security strength required for the key being generated. In the event a password encoding is simply based on ASCII, a 14-byte password is unlikely to contain sufficient entropy for most purposes. The standard set of printable characters only allows for as much as 6 bits of entropy per byte. For a 14-byte password, this yields a key that has been generated using 14 * 6 bits of entropy, giving only 84 bits of security, which is well below what is required for a key with the same level of hardness as a 112-bit one. Users are referred to Appendix A (Security Considerations) of SP 800-132 for further information on password, salt, and iteration count selection.

The iteration count value is provided by the user and should be appropriate to the way the algorithm is being used. (The memory hard augmentation of PBKDF provided by SCRYPT uses an iteration count of 1). For straight PBKDF with no memory hard support, the iteration count provided by the user should be at point of maximum cost bearable by the user carrying out the key derivation in the normal course of usage. To ensure sufficient whitening of the password in both cases, the module enforces a salt size of 128 bits in approved mode.

For users interested in introducing memory hardness as a layer on top of the PBKDF the SCrypt augmentation to PBDKF based on HMAC-SHA-256 (as described in RFC 7914) is also available in non-approved mode.

2.6.3 Rules for Setting the N and the S String in cSHAKE

To customize the output of the cSHAKE function, the cSHAKE algorithm permits the operator to input strings for the Function-Name input (N) and the Customization String (S).

The Function-Name input (N) is reserved for values specified by NIST and should only be set to the appropriate NIST specified value. Any other use of N is non-conformant.

The Customization String (S) is available to allow users to customize the cSHAKE function as they wish. The length of S is limited to the available size of a byte array in the JVM running the module.

2.6.4 Guidance for the Use of Format-Preserving Encryption

The module supports both FF1 and, in non-approved mode, FF3-1 format preserving encryption. Both are modes of AES. Table 10 shows the parameter constraints applicable to the module's implementation, as required by IG C.J.

Table 10 - SP 800-38G Format-Preserving Encryption Constraints

	FF1	FF3-1
radix	in range of 2 … 2^{16}	in range of 2 … 2^{16}
$radix^{minlen}$	$\geq$ 1,000,000	$\geq$ 1,000,000

	FF1	FF3-1
minlen	$\geq$ 2 octets	2 octets
maxlen	< 2^{32} octets	2 * *floor*($log_{radix}(2^{96})$) octets
maxTlen	$\geq$ 0 octets	8 octets (fixed)

An attempt to use the FF1 or FF3-1 without meeting the $radix^{minlen}$ constraint or by exceeding maxlen will result in an IllegalArgumentException. Note: only FF1 should be used in approved mode.

2.6.5 TLS 1.2 KDF (IG D.Q Conformance)

As indicated under CAVP certificate A4399, the module supports TLS 1.2 KDF per RFC 5246, i.e. without using the extended master secret.

2.6.6 Truncated HMACs

Approved HMAC algorithms can produce truncated versions of the specified HMAC. The right-most bits are truncated as per the NIST SP 800-107r1 (see also IG C.L and IG C.D).

2.7 RBG and Entropy

The module does not include an entropy source.

The module's use of an external Random Number Generator (RNG) is determined by the settings described in the subsections below.

Table 11 – Non-Deterministic Random Number Generation Specification

Entropy Sources	Minimum number of bits of entropy	Details
Passive Entropy	128	As per FIPS 140-3 IG 9.3.A Section 2b, a minimum of 16 bytes (128 bits) is required from the source configured for seed generation for the JVM. The entropy reader will block until the seed generator has provided the minimum number of bytes.

2.7.1 Use of External RNG

The module makes use of the JVM's configured SecureRandom entropy source to provide entropy when required. The module will request entropy as appropriate to the security strength and seeding configuration for the DRBG that is using it and for the default DRBG will request a minimum of 256 bits of entropy. In approved mode the minimum amount of entropy that can be requested by a DRBG is 112 bits. The module will wait until the *SecureRandom.generateSeed()* returns the requested amount of entropy, blocking if necessary.

The JVM's entropy source can be configured through setting the security property *securerandom.strongAlgorithms* in the JVM's java.security file.

2.7.2 Guidance for the Use of DRBGs and Configuring the JVM's Entropy Source

A user can instantiate the default Approved DRBG for the module explicitly by using *SecureRandom.getInstance*("DEFAULT", "BCFIPS"), or by using a BouncyCastleFipsProvider object instead of the provider name as appropriate. This will seed the Approved DRBG from the live entropy source of the JVM with a number of bits of entropy appropriate to the security level of the default Approved DRBG configured for the module.

The JVM's entropy source is checked according to SP 800-90B, Section 4.4 using the suggested C values for the Repetition Count Test (Section 4.4.1) and the Adaptive Proportion Test (Section 4.4.2) by default. These values can also be configured using the security property *org.bouncycastle.entropy.factors*. This property takes a comma separated list of C values: one for 4.4.1, one for 4.4.2, and a value of H. For the default, the property would be set as:

org.bouncycastle.entropy.factors: 4, 13, 8.0

in the java.security property file.

An additional option is available using the Approved Hash DRBG and the process outlined in SP 800-90A, Section 8.6.5. This can be turned on by following the instructions in Section 2.3 of the User Guide. The two DRBGs are instantiated in a chain as a "Source DRBG" to seed the "Target DRBG" in accordance with Section 7 of Draft NIST SP 800-90C, where the Target DRBG is the default Approved DRBG used by the module.

The initial seed and the subsequent reseeds for the DRBG chain come from the live entropy source configured for the JVM. The DRBG chain will reseed automatically by pausing for 20 requests (which will usually equate to 5120 bytes). An entropy gathering thread reseeds the DRBG chain when it has gathered sufficient entropy (currently 256 bits) from the live entropy source. Once reseeded, the request counter is reset and the reseed process begins again.

The "Source DRBG" in the chain is internal to the module and inaccessible to the user to ensure it is only used for generating seeds for the default Approved DRBG of the module.

The user shall ensure that the entropy source is configured per Section 2.7.1 of this Security Policy and will block, or fail, if it is unable to provide the amount of entropy requested.

2.8 Key Generation

The module performs Cryptographic Key Generation in conformance to FIPS 140-3 IG D.H. The CKG for symmetric keys and seeds used for generating asymmetric keys is performed as per Section 4 of the SP 800-133r2 (using the output of a random bit generator) and is compliant with FIPS 186-4 and SP 800-

90Ar1 for DRBG. The seed used in asymmetric key generation is the direct output of SP 800-90Ar1 DRBG.

Refer to Section 9.1 of the Security Policy for SSP generation details.

2.9 Key Establishment

The module does not perform automatic SSP establishment, it only provides the components to the calling application, which can be used in SSP establishment.

2.10 Industry Protocols

The module implements KDFs from SP 800-135r1 (Recommendation for Existing Application-Specific Key Derivation Functions). These KDFs have been validated by the CAVP and received CVL certificates (A4399). No parts of these protocols, other than the CAVP tested components, have been reviewed or tested by the CAVP and CMVP.

3 Cryptographic Module Ports and Interfaces

3.1 Ports and Interfaces

As a software cryptographic module, the module supports logical interfaces only and not physical ports. All access to the module is through the module's API. The API provides and defines the module's logical interfaces.

The module does not implement a control output interface. As a software module, the power interface is also not applicable.

The mapping of the FIPS 140-3 logical interfaces to the module is described in Table 12.

Table 12 – Ports and Interfaces

Physical Port	Logical Interface	Data That Passes Over the Port/Interface
N/A	Data Input	API input parameters – plaintext and/or ciphertext data.
N/A	Data Output	API output parameters and return values – plaintext and/or ciphertext data.
N/A	Control Input	API method calls – method calls or input parameters that specify commands and/or control data used to control the operation of the module.
N/A	Control Output	N/A, not implemented
N/A	Status Output	API output parameters and return/error codes that provide status information used to indicate the state of the module.
N/A	Power	N/A for software modules

3.2 Additional Information

All interfaces are logically separated by the module's API.

When the module performs self-tests, is in an error state, is generating keys, or performing zeroization, the module prevents all output on the logical data output interface as only the thread performing the operation has access to the data. The module is single-threaded, and in an error state, the module does not return any output data, only an error value.

4 Roles, Services, and Authentication

4.1 Authentication Methods

Not applicable.

The module does not support authentication.

4.2 Roles

The module supports two distinct operator roles, which are the User and Cryptographic Officer (CO). The cryptographic module implicitly maps the two roles to the services.

An operator is considered the owner of the thread that instantiates the module and, therefore, only one concurrent operator is allowed. The module does not support a maintenance role and/or bypass capability.

Table 13 lists all operator roles supported by the module.

Table 13 - Roles

Name	Type	Operator Type	Authentication Type	Authentication Strength
CO	Role	CO	N/A – Authentication not required for Level 1	N/A
User	Role	User	N/A – Authentication not required for Level 1	N/A

4.3 Approved Services

Table 14 lists the module services and corresponding details. The modes of SSP access shown in the table are defined as:

- G = Generate: The module generates or derives the SSP.
- R = Read: The SSP is read from the module (e.g. the SSP is output).
- E = Execute: The module uses the SSP in performing a cryptographic operation.
- W = Write: The SSP is updated, imported or written to the module.
- Z = Zeroize: The module zeroizes the SSP.

Note: The module services are the same in the approved and non-approved modes of operation. The only difference is the function(s) used (i.e. approved/allowed or non-approved/non-allowed).

Services in the module are accessed via the public APIs of the JAR file. The ability of a thread to invoke non-approved services depends on whether it has been registered with the module as approved mode only. In approved mode, no non-approved services are accessible.

Refer also to Section 6.1 and 11.2 of this Security Policy for guidance.

Table 14 – Approved Services

Name	Description	Indicator[16]	Input	Output	Approved Security Functions	Keys / SSPs	Roles	Keys/SSPs Access
Initialize Module and Run Self-Tests on Demand	The JRE will call the static constructor for self-tests on module initialization	Flag	N/A	Exception in case of failure	N/A	N/A	CO / User	N/A
Show Status	A user can call *FipsStatus.IsReady()* at any time to determine if the module is ready. *CryptoServicesRegistrar.IsInApprovedOnlyMode()* can be called to determine the approved mode of operation	Flag	N/A	Boolean	N/A	N/A	CO / User	N/A
Info Service	A user can call *DumpInfo.main()* at any time to display the module version, checksum, and status information	Flag	N/A	Module name and version, checksum, and status	N/A	N/A	CO / User	N/A
Zeroize / Power-off	The module uses the JVM garbage collector on thread termination	Flag	N/A	Shutdown indication	N/A	All SSPs	CO / User	Z

[16]Flag is accessed by calling the method *CryptoServicesRegistrar.isInApprovedOnlyMode()* - this method will return true if the thread is running in approved-only mode, false otherwise. Refer also to Section 2.4 of this Security Policy.

Name	Description	Indicator[16]	Input	Output	Approved Security Functions	Keys / SSPs	Roles	Keys/SSPs Access
Data Encryption	Used to encrypt data	Flag	Key, Plaintext	Ciphertext	AES CBC, AES CFB8, AES CFB128, AES CTR, AES ECB, AES FF1, AES OFB, AES CBC-CS1, AES CBC-CS2, AES CBC-CS3, AES CCM, AES GCM	AES Encryption Key	CO / User	E
Data Decryption	Used to decrypt data	Flag	Key, Ciphertext	Plaintext	AES CBC, AES CFB8, AES CFB128, AES CTR, AES ECB, AES FF1, AES OFB, AES CBC-CS1, AES CBC-CS2, AES CBC-CS3, AES CCM, AES GCM	AES Decryption Key	CO / User	E
MAC Calculation	Used to calculate data integrity codes with CMAC, GMAC	Flag	Key, Message	MAC	AES CMAC, AES GMAC	AES Authentication Key	CO / User	E

Name	Description	Indicator[16]	Input	Output	Approved Security Functions	Keys / SSPs	Roles	Keys/SSPs Access
Signature Generation	Used to generate digital signatures	Flag	Key, Message	Signature	DSA, ECDSA, RSA	DSA Signing Key, EC Signing Key, RSA Signing Key	CO / User	E
Signature Verification	Used to verify digital signatures	Flag	Key, Message Signature	Boolean	DSA, ECDSA, RSA	DSA Verification Key, EC Verification Key, RSA Verification Key	CO / User	E

DRBG (SP 800-90Ar1) output	Used to generate random numbers, IVs and keys	Flag	N/A	Data	Counter DRBG Hash DRBG HMAC DRBG	AES Encryption Key, AES Decryption Key, AES Authentication Key, AES Wrapping Key, DH Agreement Private Key, DH Agreement Public Key, DRBG Seed, Internal State V and C value, and DRBG Key, DSA Signing Key, EC Agreement Private Key, EC Agreement Public Key, EC Signing Key, HMAC Authentication Key, KMAC Authentication Key, RSA Signing Key, RSA Key Transport Private Key, RSA Key Transport Public Key	CO / User CO / User	G E

Name	Description	Indicator[16]	Input	Output	Approved Security Functions	Keys / SSPs	Roles	Keys/SSPs Access
						DRBG Seed, Internal State V and C value, and DRBG Key		
Message Hashing	Used to generate a message digest, SHAKE output	Flag	Message	Hash	SHS, SHA-3, SHAKE, SHA-3 Derived Functions (cSHAKE, TupleHash, ParallelHash)	N/A	CO / User	N/A
Keyed Message Hashing	Used to calculate data integrity codes with HMAC and KMAC	Flag	Key, Message	Hash	HMAC, SHA-3 Derived Functions (KMAC)	HMAC Authentication Key, KMAC Authentication Key	CO / User	E
TLS Key Derivation Function	Used to calculate a value suitable to be used for a master secret in TLS	Flag	TLS Parameters	Data	HKDF, Existing Application-Specific (TLS KDF)	TLS KDF Secret Value	CO / User	E

Name	Description	Indicator[16]	Input	Output	Approved Security Functions	Keys / SSPs	Roles	Keys/SSPs Access
SP 800-108r1 KDF	Used to calculate a value suitable to be used for a secret key	Flag	KDF Parameters	Data	KBKDF using Pseudorandom Functions	SP 800-108r1 KDF Secret Value	CO / User	E
SSH Derivation Function	Used to calculate a value suitable to be used for a secret key	Flag	SSH Parameters	Data	Existing Application-Specific (SSH KDF)	SSH KDF Secret Value	CO / User	E:
X9.63 Derivation Function	Used to calculate a value suitable to be used for a secret key	Flag	X9.63 Parameters	Data	Existing Application-Specific (X9.63 KDF)	DH Agreement Private Key, EC Agreement Private Key, RSA Signing Key X9.63 KDF Secret Value	CO / User CO / User	G E

Name	Description	Indicator[16]	Input	Output	Approved Security Functions	Keys / SSPs	Roles	Keys/SSPs Access
SP 800-56Cr2 OneStep/ TwoStep Key Derivation Function (KDM)	Used to calculate a value suitable to be used for a secret key	Flag	KDM Parameters	Data	HKDF, KDF One Step, KDF Two Step	DH Agreement Private Key, EC Agreement Private Key, RSA Signing Key SP 800-56Cr2 OneStep/TwoStep KDF Secret Value	CO / User CO / User	G E
IKEv2 Derivation Function	Used to calculate a value suitable to be used for a secret key	Flag	IKEv2 Parameters	Data	Existing Application-Specific (IKEv2 KDF)	IKEv2 KDF Secret Value	CO / User	E
SRTP Derivation Function	Used to calculate a value suitable to be used for a secret key	Flag	SRTP Parameters	Data	Existing Application-Specific (SRTP KDF)	SRTP KDF Secret Value	CO / User	E

Name	Description	Indicator[16]	Input	Output	Approved Security Functions	Keys / SSPs	Roles	Keys/SSPs Access
PBKDF	Used to generate a key using an encoding of a password and a message hash	Flag	Password, PBKDF Parameters	Data	KDF Password-Based	HMAC Authentication Key, KMAC Authentication Key HMAC Authentication Key, KMAC Authentication Key, PBKDF Secret Value	CO / User CO / User	G E

Name	Description	Indicator[16]	Input	Output	Approved Security Functions	Keys / SSPs	Roles	Keys/SSPs Access
Key Agreement Schemes	Used to calculate key agreement values	Flag	Key Agreement keys, Parameters	Data	KAS-ECC, KAS-FFC, KAS-IFC, Safe Primes	AES Encryption Key, AES Decryption Key, AES Authentication Key, AES Wrapping Key, HMAC Authentication Key, KMAC Authentication Key ——— DH Agreement Private Key, EC Agreement Private Key, RSA Key Transport Private Key	CO / User ——— CO / User	G ——— E
Key Wrapping	Used to encrypt a key value	Flag	Wrapping key, Key	Wrapped key	AES KW, AES KWP, KTS-IFC	AES Wrapping Key, HMAC Authentication Key, KMAC Authentication Key, RSA Key Transport Private Key	CO / User	E

Name	Description	Indicator[16]	Input	Output	Approved Security Functions	Keys / SSPs	Roles	Keys/SSPs Access
Key Unwrapping	Used to decrypt a key value	Flag	Unwrapping key, Wrapped key	Key	AES KW, AES KWP, KTS-IFC	AES Wrapping Key, HMAC Authentication Key, KMAC Authentication Key, RSA Key Transport Public Key	CO / User	E
Key Generation	Used to generate a key pair	Flag	Key Generation Parameters	Key Pair	RSA KeyGen, DSA KeyGen, ECDSA KeyGen, CKG	DRBG Output, DSA Signing Key, EC Signing Key, RSA Signing Key, DSA Verification Key, EC Verification Key, RSA Verification Key	CO / User	E
Key Verification	Used to verify a key pair	Flag	Key Pair	Boolean	ECDSA KeyVer	EC Signing Key, EC Verification Key	CO / User	E
Entropy Callback	Gathers entropy in a passive manner from a user-provided function	Flag	N/A	Random bits	DRBG, CKG	DRBG Seed, Internal State V and C value, and DRBG Key	CO / User	G
DRBG Health Tests	Used to perform checks of incoming entropy against Section 4.4 of SP 800-90B	Flag	N/A	N/A	DRBG	N/A	CO / User	N/A

SSP Export Operation	Returns a CSP as data that can be used for later output	Flag	SSP	Data	N/A	AES Encryption Key, AES Decryption Key, AES Authentication Key, AES Wrapping Key, DH Agreement Private Key, DH Agreement Public Key, DSA Signing Key, DSA Verification Key, EC Agreement Private Key, EC Agreement Public Key, EC Signing Key, EC Verification Key, HMAC Authentication Key, KMAC Authentication Key, RSA Signing Key, RSA Key Transport Private Key, RSA Key Transport Public Key	CO / User	R

Name	Description	Indicator[16]	Input	Output	Approved Security Functions	Keys / SSPs	Roles	Keys/SSPs Access
Utility	Miscellaneous utility functions, does not access CSPs	Flag	N/A	N/A	N/A	N/A	User	N/A

4.4 Non-Approved Services

Table 15 - Non-Approved Services

Name	Description	Indicator[17]	Algorithms Accessed	Roles
Data Encryption	Used to encrypt data	Flag	Triple-DES	CO / User
Data Decryption	Used to decrypt data	Flag	Triple-DES	CO / User
MAC Calculation	Used to calculate data integrity codes with CMAC	Flag	Triple-DES CMAC	CO / User
DRBG (SP 800-90Ar1) output	Used to generate random numbers, IVs and keys	Flag	ctrDRBG-Triple-DES	CO / User
Key Agreement Schemes	Used to calculate key agreement values	Flag	Triple-DES	CO / User
Key Wrapping	Used to encrypt a key value	Flag	Triple-DES KW	CO / User
Key Unwrapping	Used to decrypt a key value	Flag	Triple-DES KW	CO / User

5 Software/Firmware Security

5.1 Integrity Techniques

The integrity technique used by the module is HMAC-SHA-256. The integrity technique has received CAVP certificate A4399.

The integrity technique is implemented by the module itself. The HMAC of the module JAR file, excluding directories and metadata, is calculated and compared to the expected value embedded within the module's properties. If the calculated value does not match the expected value, the module raises an error and fails to load. The integrity test can be performed on demand by power cycling the host platform.

[17] Flag is accessed by calling the method *CryptoServicesRegistrar.isInApprovedOnlyMode()* - this method will return true if the thread is running in approved-only mode, false otherwise. Refer also to Section 2.4 of this Security Policy.

5.2 Initiate on Demand

Each time the module is powered up, it runs the pre-operational tests to ensure that the integrity of the module has been maintained. Self-tests are available on demand by power cycling the module.

6 Operational Environment

The module operates in a modifiable operational environment under the FIPS 140-3 definitions.

The module runs on a GPC running one of the operating systems specified in the approved operational environment list (refer to Section 2.2 of this Security Policy). Each approved operating system manages processes and threads in a logically separated manner. The module's operator is considered the owner of the calling application that instantiates the module within the process space of the Java Virtual Machine.

6.1 Configuration Settings and Restrictions

The module must be installed as described in Security Policy Section 11.1.

No specific configuration options are required for the operational environments. No security rules, settings, or restrictions to the configuration of the operational environment are needed for the module to function in a FIPS-conformant manner.

7 Physical Security

The requirements of this section are not applicable to the module. The module is a software module and does not implement any physical security mechanisms.

8 Non-Invasive Security

The requirements of this section are not applicable to the module.

9 Sensitive Security Parameter Management

All Sensitive Security Parameters (SSPs) used by the module are described in this section in Table 16. All usage of these SSPs by the module (including all SSP lifecycle states) is described in the services detailed in Section 4.3 - Approved Services. Please note that the module does not perform automatic SSP establishment, it only provides the components to the calling application, which can be used in SSP establishment.

9.1 SSPs

Table 16 - Sensitive Security Parameters (SSPs) Key Table

SSP Name / Type	Strength	Security Function & Cert. Number	Generation	Import / Export	Establishment	Storage[18]	Zeroisation	Use & related keys
AES Encryption Key	128, 192, 256 bits	AES CBC, AES CFB8, AES CFB128, AES CTR, AES ECB, AES FF1, AES OFB, AES CBC-CS1, AES CBC-CS2, AES CBC-CS3, AES CCM, AES GCM, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	AES encryption[22]

[18]The module does not provide persistent storage
[19]Key generator used in conjunction with an approved DRBG
[20]Import done via key constructor and/or factory (Electronic Entry)
[21]Export done via key recovery using *getEncoded()* method and followed by separate step to export key details as either plaintext or encrypted (Electronic Entry)

SSP Name / Type	Strength	Security Function & Cert. Number	Generation	Import / Export	Establishment	Storage[18]	Zeroisation	Use & related keys
AES Decryption Key	128, 192, 256 bits	AES CBC, AES CFB8, AES CFB128, AES CTR, AES ECB, AES FF1, AES OFB, AES CBC-CS1, AES CBC-CS2, AES CBC-CS3, AES CCM, AES GCM, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	AES decryption
AES Authentication Key	128, 192, 256 bits	AES CMAC, AES GMAC, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	AES CMAC/GMAC

[22]The AES GCM key and IV is generated randomly per IG C.H, and the Initialization Vector (IV) is a minimum of 96 bits. In the event module power is lost and restored, the consuming application must ensure that any of its AES GCM keys used for encryption or decryption are re-distributed. Refer to Section 2.6.1 of the Security Policy.

SSP Name / Type	Strength	Security Function & Cert. Number	Generation	Import / Export	Establishment	Storage[18]	Zeroisation	Use & related keys
AES Wrapping Key	128, 192, 256 bits	AES KW, AES KWP, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	AES (128/192/256) key wrapping key for KTS
DH Agreement Private Key	112, 128, 152, 176, 200 bits	KAS-FFC, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	Diffie-Hellman (ffdhe and MODP) key agreement May be paired with DH Agreement Public Key
DH Agreement Public Key	112, 128, 152, 176, 200 bits	KAS-FFC, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	Not zeroized, public key value known outside of module	Diffie-Hellman (ffdhe and MODP) key agreement May be paired with DH Agreement Private Key

SSP Name / Type	Strength	Security Function & Cert. Number	Generation	Import / Export	Establishment	Storage[18]	Zeroisation	Use & related keys
DSA Signing Key	112, 128 bits	DSA Signature Generation, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	DSA signature generation May be paired with DSA Verification Key
DSA Verification Key	80, 112, 128 bits	DSA Signature Verification, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	Not zeroized, public key value known outside of module	DSA signature verification May be paired with DSA Signing Key
EC Agreement Private Key	112, 128, 192, 256 bits	KAS-ECC, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	EC key agreement May be paired with EC Agreement Public Key

SSP Name / Type	Strength	Security Function & Cert. Number	Generation	Import / Export	Establishment	Storage[18]	Zeroisation	Use & related keys
EC Agreement Public Key	112, 128, 192, 256 bits	KAS-ECC, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	Not zeroized, public key value known outside of module	EC key agreement May be paired with EC Agreement Private Key
EC Signing Key	112, 128, 192, 256 bits	ECDSA Signature Generation, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	ECDSA signature generation. May be paired with EC Verification Key
EC Verification Key	112, 128, 192, 256 bits	ECDSA Signature Verification, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	Not zeroized, public key value known outside of module	ECDSA signature verification. May be paired with EC Signing Key

SSP Name / Type	Strength	Security Function & Cert. Number	Generation	Import / Export	Establishment	Storage[18]	Zeroisation	Use & related keys
HMAC Authentication Key	112-256 bits	HMAC-SHA-1, HMAC-SHA-2, HMAC-SHA-3, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	Keyed-Hash Calculation
KMAC Authentication Key	112-256 bits	KMAC, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	Keyed-Hash Calculation
RSA Signing Key	112, 128, 152 bits	RSA Signature Generation, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	RSA signature generation May be paired with RSA Verification Key
RSA Verification Key	80, 112, 128, 152 bits	RSA Signature Verification, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	Not zeroized, public key value known outside of module	RSA signature verification May be paired with RSA Signing Key

SSP Name / Type	Strength	Security Function & Cert. Number	Generation	Import / Export	Establishment	Storage[18]	Zeroisation	Use & related keys
RSA Key Transport Private Key[23]	112, 128, 152 bits	KTS-IFC, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	RSA key transport and decryption May be paired with RSA Public Key Transport Key
RSA Key Transport Public Key[23]	112, 128, 152 bits	KTS-IFC, CKG A4399	DRBG[19]	Import[20], Export[21]	N/A	N/A	Not zeroized, public key value known outside of module	RSA key transport May be paired with RSA Key Transport Private Key
IKEv2 KDF Secret Value	112, 128, 192, 256 bits	KDF IKEv2 A4399	Generated as output of an IKEv2 agreement scheme	N/A	N/A	N/A	*destroy()* service call or host platform power cycle	Key Derivation
PBKDF Secret Value	112-256 bits	PBKDF A4399	Generated as output of a PBE key and a PRF	N/A	N/A	N/A	*destroy()* service call or host platform power cycle	Key Derivation

[23]RSA key transport using PKCS#1 1.5 padding is deprecated through 2023 and disallowed after 2023.

SSP Name / Type	Strength	Security Function & Cert. Number	Generation	Import / Export	Establishment	Storage[18]	Zeroisation	Use & related keys
SP 800-56Cr2 OneStep/ TwoStep KDF Secret Value	112, 128, 192, 256 bits	KDA OneStep SP 800-56Cr2, KDA TwoStep SP 800-56Cr2 A4399	Generated as output of an agreement scheme	N/A	N/A	N/A	*destroy()* service call or host platform power cycle	Key Derivation
SP 800-108r1 KDF Secret Value	112, 128, 192, 256 bits	KDF SP 800-108 A4399	Generated as output of an agreement scheme	N/A	N/A	N/A	*destroy()* service call or host platform power cycle	Key Derivation
SRTP KDF Secret Value	128, 192, 256 bits	KDF SRTP A4399	Generated as output of an SRTP agreement scheme	N/A	N/A	N/A	*destroy()* service call or host platform power cycle	Key Derivation
SSH KDF Secret Value	80, 112, 128, 192, 256 bits	KDF SSH A4399	Generated as output of an SSH agreement scheme	N/A	N/A	N/A	*destroy()* service call or host platform power cycle	Key Derivation

SSP Name / Type	Strength	Security Function & Cert. Number	Generation	Import / Export	Establishment	Storage[18]	Zeroisation	Use & related keys
TLS Premaster Secret Value	384 bits	KDF TLS A4399	Protocol version (2 bytes) and 46 bytes from a DRBG[19]	Import[20], Export[21]	N/A	N/A	*destroy()* service call or host platform power cycle	Used to derive keys using TLS KDF
TLS KDF Secret Value	112, 128, 192, 256 bits	KDF TLS A4399	Generated as output of TLS agreement scheme	N/A	N/A	N/A	*destroy()* service call or host platform power cycle	Key Derivation
X9.63 KDF Secret Value	112, 128, 192, 256 bits	KDF ANS 9.63 A4399	Generated as output of an agreement scheme	N/A	N/A	N/A	*destroy()* service call or host platform power cycle	Key Derivation
Entropy Input String	>128 bits	N/A	N/A	Obtained from the entropy source	N/A	N/A	*destroy()* service call or host platform power cycle	Random Number Generation
CTR DRBG Seed	128, 192, 256 bits	N/A	N/A	From external entropy source	N/A	N/A	Immediately after use or host platform power cycle	Internal use
CTR DRBG V Value	128 bits	N/A	From seed value	N/A	N/A	N/A	*reseed()* service call or host platform power cycle	Internal use

SSP Name / Type	Strength	Security Function & Cert. Number	Generation	Import / Export	Establishment	Storage[18]	Zeroisation	Use & related keys
CTR DRBG Key	128, 192, 256 bits	N/A	From DRBG V value	N/A	N/A	N/A	*reseed()* service call or host platform power cycle	Internal use
Hash DRBG Seed	112, 128, 192, 256 bits	N/A	N/A	From external entropy source	N/A	N/A	Immediately after use or host platform power cycle	Internal use
Hash DRBG V Value	112, 128, 192, 256 bits	N/A	From seed value	N/A	N/A	N/A	*reseed()* service call or host platform power cycle	Internal use
Hash DRBG C Value	112, 128, 192, 256 bits	N/A	From DRBG V value	N/A	N/A	N/A	*reseed()* service call or host platform power cycle	Internal use
HMAC DRBG Seed	112, 128, 192, 256 bits	N/A	N/A	From external entropy source	N/A	N/A	Immediately after use or host platform power cycle	Internal use
HMAC DRBG V Value	112, 128, 192, 256 bits	N/A	From seed value	N/A	N/A	N/A	*reseed()* service call or host platform power cycle	Internal use

SSP Name / Type	Strength	Security Function & Cert. Number	Generation	Import / Export	Establishment	Storage[18]	Zeroisation	Use & related keys
HMAC DRBG Key	112, 128, 192, 256 bits	N/A	From DRBG V value	N/A	N/A	N/A	*reseed()* service call or host platform power cycle	Internal use
DRBG Output	128, 192, 256 bits	N/A	DRBG	N/A	N/A	N/A	*destroy()* service call or host platform power cycle	Used as seed for asymmetric key generation or for symmetric key generation

10 Self-Tests

Cryptographic Algorithm Self-Tests (CASTs) are performed prior to the first use of services related to the test target. CASTs also run periodically on service invocation.

Pairwise Consistency Tests (PCTs) are performed on the corresponding key pairs.

10.1 Pre-Operational Self-Tests

Each time the module is powered up, it performs the pre-operational self-tests to confirm that sensitive data has not been damaged.

The pre-operational tests include the software integrity test, which verifies the module using HMAC-SHA-256. Pre-operational tests also include the HMAC and SHS CASTs that are run prior to the software integrity test to ensure the correctness of the HMAC used. Pre-operational self-tests are available on demand by power cycling the module.

10.2 Conditional Self-Tests

The module performs conditional self-tests when the conditions specified for cryptographic algorithm self-test and pair-wise consistency tests occur. The self-tests implemented are specified below.

Table 17 – Conditional Algorithm Self-Tests

Test Target	Description
AES ECB	Encryption KAT (128 bits)
AES ECB	Decryption KAT (128 bits)
AES CCM	Encryption KAT (128 bits)
AES CCM	Decryption KAT (128 bits)
AES CMAC	Generation KAT (128 bits)
AES CMAC	Verification KAT (128 bits)
AES GCM	Encrypt KAT (128 bits)
AES GCM	Decrypt KAT (128 bits)
HASH DRBG	SHA2-256 KAT (Health Tests: Generate, Reseed, Instantiate functions per Section 11.3 of SP 800-90Ar1)
HMAC DRBG	HMAC-SHA2-256 KAT (Health Tests: Generate, Reseed, Instantiate functions per Section 11.3 of SP 800-90Ar1)
CTR DRBG	AES CTR 256 bits KAT (Health Tests: Generate, Reseed, Instantiate functions per Section 11.3 of SP 800-90Ar1)
DSA	Signature Generation KAT (2048 bits)
DSA	Signature Verification KAT (2048 bits)

Test Target	Description
ECDSA	Signature Generation KAT (P-256)
ECDSA	Signature Verification KAT (P-256)
HMAC-SHA2-256	HMAC-SHA2-256 KAT
HMAC-SHA2-512	HMAC-SHA2-512 KAT
HMAC-SHA3-256	HMAC-SHA3-256 KAT
KAS-ECC	Primitive "Z" Computation KAT (P-256)
KAS-ECC	Primitive "Z" Computation KAT (B-233)
KAS-FFC	Primitive "Z" Computation KAT (ffdhe2048)
KBKDF	KBKDF KAT (Counter, Feedback, Double Pipeline)
KDA OneStep	KDA OneStep KAT
KDA TwoStep	KDA TwoStep KAT
PBKDF	PBKDF KAT (HMAC-SHA2-256)
RSA	Signature Generation KAT (2048 bits)
RSA	Signature Verification KAT (2048 bits)
RSA Encryption	RSA Encryption KAT SP 800-56Br2 (2048 bits)
RSA Decryption	RSA Decryption KAT SP 800-56Br2 (2048 bits)
SHA-1	SHA-1 KAT
SHA2-256	SHA2-256 KAT
SHA2-512	SHA2-512 KAT
SHA-3	SHA-3 KAT (cSHAKE-128)
SHAKE256	SHAKE256 KAT
ANS 9.63 KDF	ANS 9.63 KDF KAT
IKEv2 KDF	IKEv2 KDF KAT
SNMP KDF	SNMP KDF KAT
SRTP KDF	SRTP KDF KAT
SSH KDF	SSH KDF KAT
TLS 1.0 KDF	TLS 1.0 KDF KAT
TLS 1.1 KDF	TLS 1.1 KDF KAT
TLS 1.2 KDF	TLS 1.2 KDF KAT

Table 18 – Pairwise Consistency Tests

Test Target	Description
DH	DH Pairwise Consistency Test
DSA	DSA Pairwise Consistency Test
EC DH	EC DH Pairwise Consistency Test
ECDSA	ECDSA Pairwise Consistency Test
RSA	RSA Pairwise Consistency Test

10.3 Error States

If any of the above-mentioned self-tests fail, the module enters an error state called "Hard Error" state. Upon entering the error state, the module outputs status by way of an exception. An example exception for AES Encryption failure is:

"Failed self-test on encryption: AES"

The module can be recovered by power cycling, which results in execution of pre-operational self-tests and conditional cryptographic algorithm self-tests. If the tests pass, then the module will be available for use.

10.4 Operator Initiation of Self-Tests

Each time the module is powered up, it runs the pre-operational tests to ensure that the integrity of the module has been maintained. Pre-operational self-tests are available on demand by power cycling the module. Initial CAST self-tests are available on demand by power cycling the module and then invoking the service related to the test target.

11 Life-Cycle Assurance

11.1 Installation, Initialization, and Startup Procedures

The module exists as part of the running JVM, and as such:

- Secure installation of the module requires the use of the unchanged jar to be loaded into a JVM via either the class-path or the module-path as appropriate to the JVM and its usage.
- Initialization of the module will occur on startup of the module by the JVM. The user can trigger initialization by attempting to invoke any service in the module or simply calling *FipsStatus.isReady()* which will only return true if the module has been successfully initialized.
- Once the JVM has loaded the module and the module has been initialized, the startup phase is over, and the module is able to provide services.
- Operation of the module consists of calling the various APIs providing services. The module code will make use of the current thread for performing any required CASTs and health tests and then provide a service object to the user, capable of performing the requested service.

A User Guide is provided to operators of the module.

11.2 Basic Guidance

The JAR file representing the module needs to be installed in a JVM's class path in a manner appropriate to its use in applications running on the JVM.

Functionality in the module is provided in two ways. At the lowest level there are distinct classes that provide access to the approved and non-approved services provided by the module. A more abstract level of access can also be gained by using strings providing operation names passed into the module's Java cryptography provider through the APIs described in the Java Cryptography Architecture (JCA) and the Java Cryptography Extension (JCE).

When the module is used in approved mode, classes providing implementations of algorithms that are not approved or allowed are explicitly disabled.

SSPs such as private and secret keys implement the *Destroyable* interface. Where appropriate these SSPs can be zeroized on demand by invoking the *destroy()* method. The return of the *destroy()* method indicates that the zeroization is complete.

11.3 Use of the JVM with a Java SecurityManager

If the underlying JVM is running with a Java SecurityManager installed, the module will be running in approved mode with secret and private key export disabled.

11.3.1 Additional Enforcement with a Java SecurityManager

In the presence of a Java SecurityManager approved mode services specific to a context, such as DSA and ECDSA for use in TLS, require specific policy permissions to be configured in the JVM configuration by the Cryptographic Officer or User. The SecurityManager can also be used to restrict the ability of particular code bases to examine CSPs.

In the absence of a Java SecurityManager specific services related to protocols such as TLS are available, however must only be used in relation to those protocols.

11.3.2 Permissions for Java SecurityManager

Use of the module with a Java SecurityManager requires the setting of some basic permissions to allow the module HMAC-SHA-256 software integrity test to take place as well as to allow the module itself to examine secret and private keys. The basic permissions required for the module to operate correctly with a Java SecurityManager are indicated by the **Required** column of Table 19.

Table 19 - Available Java Permissions for SecurityManager

Permission	Settings	Required	Usage
RuntimePermission	*getProtectionDomain*	Yes	Allows checksum to be carried out on JAR.
RuntimePermission	*accessDeclaredMembers*	Yes	Allows use of reflection API within the provider.
PropertyPermission	*java.runtime.name, read*	No	Only if configuration properties are used.
SecurityPermission	*putProviderProperty.BCFIPS*	No	Only if provider installed during execution.
CryptoServicesPermission	*unapprovedModeEnabled*	No	Only if non-approved mode algorithms required.
CryptoServicesPermission	*changeToApprovedModeEnabled*	No	Only if threads allowed to change modes.
CryptoServicesPermission	*exportSecretKey*	No	To allow export of secret keys only.
CryptoServicesPermission	*exportPrivateKey*	No	To allow export of private keys only.
CryptoServicesPermission	*exportKeys*	Yes	Required to be applied for the module itself. Optional for any other codebase.
CryptoServicesPermission	*tlsNullDigestEnabled*	No	Only required for TLS digest calculations.
CryptoServicesPermission	*tlsPKCS15KeyWrapEnabled*	No	Only required if TLS is used with RSA encryption.
CryptoServicesPermission	*tlsAlgorithmsEnabled*	No	Enables both NullDigest and PKCS15KeyWrap.

Permission	Settings	Required	Usage
CryptoServicesPermission	*defaultRandomConfig*	No	Allows setting of default SecureRandom.
CryptoServicesPermission	*threadLocalConfig*	No	Required to set a thread local property in the CryptoServicesRegistrar.
CryptoServicesPermission	*globalConfig*	No	Required to set a global property in the CryptoServicesRegistrar.

11.4 Design and Rules

The module design corresponds to the module security rules. This section documents the security rules enforced by the cryptographic module to implement the security requirements of this FIPS 140-3 Level 1 module.

1. The module provides two distinct operator roles: User and Cryptographic Officer.
2. The module does not provide authentication.
3. The operator may command the module to perform the self-tests by cycling power or resetting the module.
4. Self-tests do not require any operator action.
5. Data output is inhibited during self-tests, zeroization, and error states. Output related to keys and their use is inhibited until the key concerned has been fully generated.
6. Status information does not contain CSPs or sensitive data that if misused could lead to a compromise of the module.
7. There are no restrictions on which keys or CSPs are zeroized by the zeroization service.
8. The module does not support concurrent operators.
9. The module does not have any external input/output devices used for entry/output of data.
10. The module does not enter or output plaintext CSPs from the module's physical boundary.
11. The module does not output intermediate key values.

11.4.1 Mode of Operation Rules

When the module is used within the context of Java Security Manager or the system/security property *org.bouncycastle.fips.approved_only* is set to true, the module will start in approved mode and non-approved services are not accessible in this mode. When the module is not used within the context of Java Security Manager, the module will start in non-approved mode by default. Refer to Security Policy Section 2.4 for additional details.

11.4.1.1 From Non-Approved Mode to Approved Mode

The transition from non-approved mode to approved mode is a combination of granted permission (a) and request to change mode (b):

a) *org.bouncycastle.crypto.CryptoServicesPermission "changeToApprovedModeEnabled"*

b) *CryptoServicesRegistrar.setApprovedMode(true)*

The CSPs made available in non-approved mode will not be accessible once the thread transitions into approved mode. The CSPs generated using the non-approved mode cannot be passed or shared with algorithms operating in approved mode, and vice-versa. This is done by an indicator within the class (object) instantiating the key that the key was created in an approved mode or non-approved mode.

Any attempt by a thread within the module to use the key in an opposite mode will result in an exception being generated by the module. For example, if an RSA private key has been created in either approved or non-approved mode, then any request to access that key will first need to confirm if the thread making the request is in the same mode.

11.4.1.2 From Approved Mode to Non-Approved Mode

The module cannot transition from approved mode to non-approved mode. To initiate the module in non-approved mode, either it should not be used in the context of Java Security Manager, or the module should have the permission *org.bouncycastle.crypto.CryptoServicesPermission unapprovedModeEnabled* granted by the Java Security Manager

11.5 Vulnerabilities

Vulnerabilities found in the module will be reported on the National Vulnerability Database, located at the following link: https://nvd.nist.gov/

Researchers and users are encouraged to report any security related concerns to support@safelogic.com.

12 Mitigation of Other Attacks

The module implements basic protections to mitigate against timing-based attacks against its internal implementations. There are two countermeasures used.

The first countermeasure is Constant Time Comparisons, which protect the digest and integrity algorithms by strictly avoiding “fast fail” comparison of MACs, signatures, and digests so the time taken to compare a MAC, signature, or digest is constant regardless of whether the comparison passes or fails.

The second countermeasure is made up of Numeric Blinding and decryption/signing verification which both protect the RSA algorithm.

Numeric Blinding prevents timing attacks against RSA decryption and signing by providing a random input into the operation which is subsequently eliminated when the result is produced. The random input makes it impossible for a third party observing the private key operation to attempt a timing attack on the operation as they do not have knowledge of the random input and consequently the time taken for the operation tells them nothing about the private value of the RSA key.

Decryption/signing verification is carried out by calculating a primitive encryption or signature verification operation after a corresponding decryption or signing operation before the result of the decryption or signing operation is returned. The purpose of this is to protect against Lenstra's CRT attack by verifying the correctness of the private key calculations involved. Lenstra's CRT attack takes advantage of undetected errors in the use of RSA private keys with CRT values and, if exploitable, can be used to discover the private value of the RSA key.

Appendix: References and Acronyms

The following standards are referred to in this Security Policy.

Table 20 - References

Abbreviation	Full Specification Name
ANSI X9.31	X9.31-1998, Digital Signatures using Reversible Public Key Cryptography for the Financial Services Industry (rDSA), September 9, 1998
FIPS 140-3	Security Requirements for Cryptographic modules, March 22, 2019
FIPS 180-4	Secure Hash Standard (SHS)
FIPS 186-2	Digital Signature Standard (DSS)
FIPS 186-4	Digital Signature Standard (DSS)
FIPS 197	Advanced Encryption Standard
FIPS 198-1	The Keyed-Hash Message Authentication Code (HMAC)
FIPS 202	SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions
IG	Implementation Guidance for FIPS PUB 140-3 and the Cryptographic Module Validation Program
PKCS#1 v2.1	RSA Cryptography Standard
PKCS#5	Password-Based Cryptography Standard
PKCS#12	Personal Information Exchange Syntax Standard -Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher
SP 800-38A	Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode
SP 800-38B	Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication
SP 800-38C	Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality
SP 800-38D	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
SP 800-38F	Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping
SP 800-38G	Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption
SP 800-56Ar3	Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography
SP 800-56Br2	Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography
SP 800-56Cr2	Recommendation for Key Derivation through Extraction-then-Expansion
SP 800-67r2	Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher
SP 800-89	Recommendation for Obtaining Assurances for Digital Signature Applications
SP 800-90A	Recommendation for Random Number Generation Using Deterministic Random Bit Generators
SP 800-90B	Recommendation for the Entropy Sources Used for Random Bit Generation

Abbreviation	Full Specification Name
SP 800-108r1	Recommendation for Key Derivation Using Pseudorandom Functions
SP 800-131A	Transitioning the Use of Cryptographic Algorithms and Key Lengths
SP 800-132	Recommendation for Password-Based Key Derivation
SP 800-133r2	Recommendation for Cryptographic Key Generation
SP 800-135r1	Recommendation for Existing Application – Specific Key Derivation Functions
SP 800-185	SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash

The following acronyms are used in this Security Policy.

Table 21 - Acronyms

Acronym	Definition
AES	Advanced Encryption Standard
API	Application Programming Interface
CAST	Cryptographic Algorithm Self-Test
CBC	Cipher-Block Chaining
CCM	Counter with CBC-MAC
CCCS	Canadian Centre for Cyber Security
CDH	Computational Diffie-Hellman
CFB	Cipher Feedback Mode
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CO	Cryptographic Officer
CPU	Central Processing Unit
CS	Ciphertext Stealing
CSP	Critical Security Parameter
CTR	Counter Mode
CVL	Component Validation List
DES	Data Encryption Standard
DH	Diffie-Hellman
DRAM	Dynamic Random Access Memory
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
DSTU4145	Ukrainian DSTU-4145-2002 Elliptic Curve Scheme
EC	Elliptic Curve
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EdDSA	Edwards Curve DSA using Ed25519, Ed448
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
GMAC	Galois Message Authentication Code
GOST	Gosudarstvennyi Standard Soyuza SSR/Government Standard of the Union of Soviet Socialist Republics
GPC	General Purpose Computer
HMAC	(Keyed) Hashed Message Authentication Code
IG	Implementation Guidance, see References
IV	Initialization Vector
JAR	Java ARchive

Acronym	Definition
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
KAS	Key Agreement Scheme
KAT	Known Answer Test
KDF	Key Derivation Function
KW	Key Wrap
KWP	Key Wrap with Padding
KMAC	KECCAK Message Authentication Code
MAC	Message Authentication Code
MD5	Message Digest algorithm MD5
N/A	Not Applicable
OCB	Offset Codebook Mode
OFB	Output Feedback
OS	Operating System
PBKDF	Password-Based Key Derivation Function
PKCS	Public Key Cryptography Standards
PQG	Diffie-Hellman Parameters P, Q and G
RC	Rivest Cipher, Ron’s Code
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
RSA	Rivest Shamir Adleman
SHA	Secure Hash Algorithm
SSP	Sensitive Security Parameter
TLS	Transport Layer Security
USB	Universal Serial Bus
XDH	Edwards Curve Diffie-Hellman using X25519, X448
XOF	Extendable-Output Function