Legion of the Bouncy Castle Inc.

BC-FJA (Bouncy Castle FIPS Java API)

Non-Proprietary FIPS 140-3 Cryptographic Module Security Policy

Software Version: 2.1.0
Date: 01/07/2025

Legion of the Bouncy Castle Inc.
(ABN 84 166 338 567)
https://www.bouncycastle.org

Table of Contents

List of Tables

List of Figures

1 General

This document defines the Security Policy for the Legion of the Bouncy Castle Inc.'s BC-FJA (Bouncy Castle FIPS Java API) Module, hereafter denoted the Module. The Module is a cryptographic library and has a Multi-Chip Stand Alone embodiment. The Module meets FIPS 140-3 overall Level 1 requirements. The SW version is 2.1.0.

The FIPS 140-3 security levels for the Module are given in Table 1as follows:

ISO/IEC 24759 Section 6. Number	Security Requirement	Security Level
1	General	1
2	Cryptographic Module Specification	1
3	Cryptographic Module Ports and Interfaces	1
4	Roles, Services, and Authentication	1
5	Software/Firmware Security	1
6	Operational Environment	1
7	Physical Security	N/A
8	Non-invasive Security	N/A
9	Sensitive Security Parameter Management	1
10	Self-Tests	1
11	Life-cycle Assurance	1
12	Mitigation of Other Attacks	1

Table 1: Security Levels

1.1 Confirming the Module Checksum, Functionality, and Versioning

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

```
java -cp bc-fips-2.1.0.jar org.bouncycastle.util.DumpInfo
```

which should display:

```
Version Info: BouncyCastle Security Provider (FIPS edition) v2.1.0
FIPS Ready Status: READY
```

```
Module SHA2-256 HMAC:
941ebff8db149f871fbbeaf90269c19453b1e9d3777541fda1c0cf9132b426ce
```

Indicating the jar represents the software release BC-FJA 2.1.0, that it has successfully passed all its startup tests, and that the software release is confirmed to have a HMAC of:

```
941ebff8db149f871fbbeaf90269c19453b1e9d3777541fda1c0cf9132b426ce
```

1.1.1 Additional Information Displayed if Native Support Available

Where native support is available `DumpInfo` will also provide what native facilities are supported. The module will make use of AES and SHA2 acceleration where available in addition to accepting entropy directly from the underlying hardware if possible. On a platform with native support turned on you will see something like:

```
Version Info: BouncyCastle Security Provider (FIPS edition) v2.1.0
FIPS Ready Status: READY
Native Ready Status: READY
Native Variant: avx
Native Build Date: 2023-07-13T06:08:49
Native Support: AES/CBC AES/CFB AES/CTR AES/ECB AES/GCM DRBG NRBG
Module SHA2-256 HMAC:
941ebff8db149f871fbbeaf90269c19453b1e9d3777541fda1c0cf9132b426ce
```

Note that, for validation purposes, the native component of the module is included in the module checksum regardless of whether it is enabled or not.

2 Cryptographic Module Specification

The Module is intended for use by US Federal agencies and other markets that require a FIPS 140-3 validated Cryptographic Library. The Module is of type software and the module has a Multi-Chip Stand Alone embodiment; the cryptographic boundary is the Java Archive (JAR) file, *bc-fips-2.1.0.jar*.

This module is the only software component within the Cryptographic Boundary and the only software component that carries out cryptographic functions covered by FIPS 140-3. Figure 1 shows the logical relationship of the cryptographic module to the other software and hardware components of the computer. The BC classes are executed on the Java Virtual Machine (JVM) using the classes of the Java Runtime Environment (JRE) with the performance being hardware assisted through augmentation where possible by different native functions in the contained runtime libraries. The JVM is the interface to the computer's Operating System (OS) that is the interface to the various physical components of the computer.

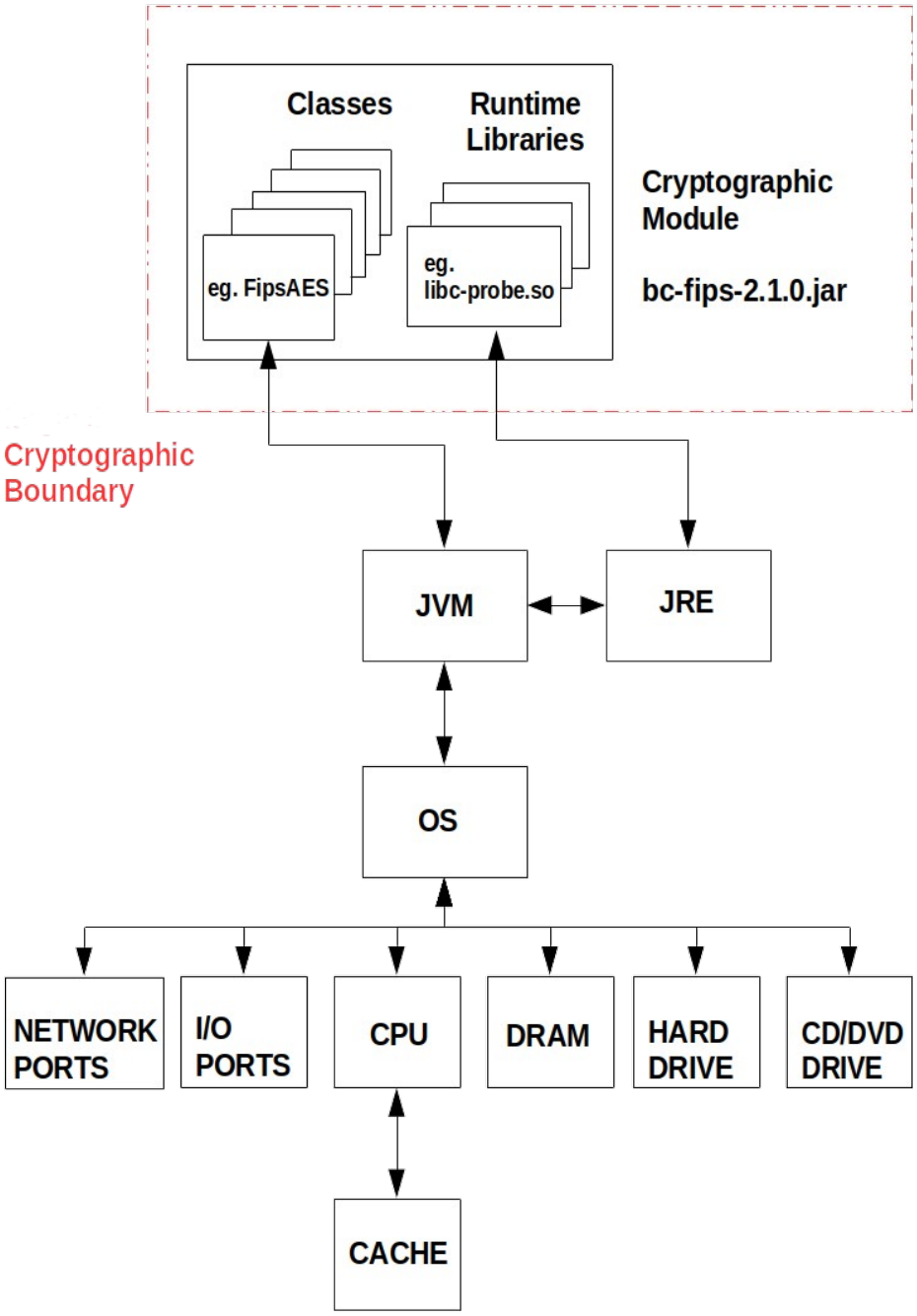

Figure 1: Cryptographic Boundary

The cryptographic module was tested on the following operational environments on the general-purpose computer (GPC) platforms detailed in Table 2, which is also the TOEPP (Tested Operational Environment's Physical Perimeter) of the module.

#	Operating System	Hardware Platform	Processor	PAA/Acceleration
1	Java SE Runtime Environment v8 (1.8) on	Intel NUC 11 Pro Board	11th Gen Intel Core i7	None

	Ubuntu 22.04 LTS			
2	Java SE Runtime Environment v8 (1.8) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (AVX)
3	Java SE Runtime Environment v8 (1.8) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (VAES)
4	Java SE Runtime Environment v8 (1.8) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (VAESF)
5	Java SE Runtime Environment v11 (1.11) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	None
6	Java SE Runtime Environment v11 (1.11) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (AVX)
7	Java SE Runtime Environment v11 (1.11) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (VAES)
8	Java SE Runtime Environment v11 (1.11) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (VAESF)
9	Java SE Runtime Environment v17 (1.17) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	None
10	Java SE Runtime Environment v17 (1.17) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (AVX)
11	Java SE Runtime Environment v17 (1.17) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (VAES)
12	Java SE Runtime Environment v17 (1.17) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (VAESF)
13	Java SE Runtime Environment v21 (1.21) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	None

14	Java SE Runtime Environment v21 (1.21) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (AVX)
15	Java SE Runtime Environment v21 (1.21) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (VAES)
16	Java SE Runtime Environment v21 (1.21) on Ubuntu 22.04 LTS	Intel NUC 11 Pro Board	11th Gen Intel Core i7	With PAA (VAESF)

Table 2: Tested Operational Environments

The cryptographic module will remain compliant with the FIPS 140-3 validation when operating on any general purpose computer (GPC) provided that:

1) No source code has been modified.
2) The GPC uses the specified single-user platform, or another compatible single-user platform such as one of the Java SE Runtime Environments listed on any of the following:

#	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 (1.21) 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 (1.21) 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 (1.21) 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 (1.21) 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 (1.21) 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 (1.21) 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 (1.21) 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 v17 (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 (1.21) 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 (1.21) with Linux SUSE	Generic 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 (1.21) 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 (1.21) 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 (1.21) 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 (1.21) 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 (1.21) 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 (1.21) 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 (1.21) 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 (1.21) 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 (1.21) with Ubuntu	Generic hardware platform with Intel Cascade Lakes
69	Java SE Runtime Environment v17 (1.17) with Ubuntu	Generic hardware platform with Intel Sapphire Rapids
70	Java SE Runtime Environment v21 (1.21) 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 (1.21) with ClevOS	Generic hardware platform with Intel Cascade Lake
73	Java SE Runtime Environment v17 (1.17) with ClevOS	Generic hardware platform with Intel Sapphire Rapids
74	Java SE Runtime Environment v21 (1.21) 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 (1.21) 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 (1.21) with ClevOS	Generic hardware platform with Intel Broadwell

Table 3: Vendor Affirmed Operational Environments

For the avoidance of doubt, it is hereby stated that 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.

The Module implements the Approved and Non-Approved but Allowed cryptographic functions with no security claimed listed in Table 4 and Table 5 below. 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. The Module supports both Approved and Non-Approved mode of operation. Please see Section 6.3 for configuration of the Module in Approved mode of operation. Please see Section 6.1 for initialization steps.

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Sizes(s)/Key Strength(s)	Use/Function
A4270	AES [FIPS 197, SP 800-38A], AES-FF1 Format Preserving Encryption [SP 800-38G]	ECB, CBC, OFB, CFB8, CFB128, CTR	Key sizes: 128, 192, 256 bits	Encryption, Decryption
A4270	AES-CBC Ciphertext Stealing (CS) [Addendum to SP 800-38A, Oct 2010]	CBC-CS1, CBC-CS2, CBC-CS3	Key sizes: 128, 192, 256 bits	Encryption, Decryption
A4270	CCM [SP 800-38C]	N/A	Key sizes: 128, 192, 256 bits	Generation, Authentication
A4270	CMAC [SP 800-38B]	AES	Key sizes: AES with 128, 192, 256 bits	Generation, Authentication
A4270	GCM/GMAC[1] [SP 800-38D]	N/A	Key sizes: 128, 192, 256 bits	Generation, Authentication
A4270	Counter DRBG [SP 800-90Ar1]	N/A	AES-128, AES-192, AES-256	Random Number Generation
A4270	Hash DRBG [SP 800-90Ar1]	N/A	SHA sizes: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256	Random Number Generation
A4270	HMAC DRBG [SP 800-90Ar1]	N/A	SHA sizes: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256	Random Number Generation

1 GCM encryption with an internally generated IV, see section 2.2 concerning external IVs. IV generation is compliant with IG C.H.

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Sizes(s)/Key Strength(s)	Use/Function
A4270	DSA[2] [FIPS 186-5]	N/A	Key sizes: 1024, 2048, 3072 bits (1024 only for SigVer)	PQG Generation, PQG Verification, Key Pair Generation, Signature Generation, Signature Verification
A4270	ECDSA [FIPS 186-5]	N/A	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 * Curves only used for Signature Verification and Public Key Validation	Public Key Generation, Signature Generation, Signature Verification, Public Key Validation
A4270	EdDSA [FIPS 186-5]	N/A	Curves/Key sizes: Ed25519, Ed448	Public Key Generation, Signature Generation, Signature Verification, Public Key Validation
A4270	KDA-HKDF [SP 800-56C, Rev 2]	N/A	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	Key Derivation

2 DSA signature generation with SHA-1 is only for use with protocols.

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Sizes(s)/Key Strength(s)	Use/Function
A4270	HMAC [FIPS 198-1]	N/A	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	Generation, Authentication
A4270	KAS-FFC[3] [SP 800-56A-rev3]	N/A	Domain Parameter Generation Methods/Scheme: 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 provide between 112 and 200 bits of encryption strength	Key Agreement
A4270	KAS-ECC[3] [SP 800-56A-rev3]	N/A	Curves: 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 provide between 112 and 256 bits of encryption strength	Key Agreement

3 Keys are not established directly into the module using the key agreement algorithms. KAS-FFC and KAS-ECC are approved key agreement methods per FIPS 140-3 IG D.F.

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Sizes(s)/Key Strength(s)	Use/Function
A4270	KDA, One Step [SP 800-56C-rev2]	N/A	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	Key Derivation
A4270	KDA, Two Step [SP 800-56C-rev2]	N/A	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, KMAC-128, KMAC-256	Key Derivation
CVL A4270	KDF, Existing Application-Specific[4] [SP 800-135-rev1]	N/A	TLS v1.0/1.1 KDF SHA sizes: SHA2-256 , SHA2-384, SHA2-512	Key Derivation
CVL A4270	KDF, Existing Application-Specific[4] [SP 800-135-rev1]	N/A	TLS 1.2 KDF SHA sizes: SHA2-256, SHA2-384, SHA2-512	Key Derivation
CVL A4270	KDF, Existing Application-Specific[4] [SP 800-135-rev1]	N/A	SSH KDF SHA sizes: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	Key Derivation

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

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Sizes(s)/Key Strength(s)	Use/Function
CVL A4270	KDF, Existing Application-Specific[4] [SP 800-135-rev1]	N/A	X9.63 KDF SHA sizes: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	Key Derivation Can be used along with KAS-SSC
CVL A4270	KDF, Existing Application-Specific[4] [SP 800-135-rev1]	N/A	IKEv2 KDF SHA sizes: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	Key Derivation
CVL A4270	KDF, Existing Application-Specific[4] [SP 800-135-rev1]	N/A	SRTP KDF	Key Derivation
A4270	KDF, Password-Based [SP 800-132]	N/A	Options: PBKDF with Option 1a Types: HMAC-based KDF using SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	Key Derivation
A4270	KDF, using Pseudorandom Functions[5] [SP 800-108]	Counter Mode, Feedback Mode, Double-Pipeline Iteration Mode	Types: CMAC-based KBKDF with AES, HMAC-based KBKDF with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	Key Derivation
A4270	Key Wrapping Using Block Ciphers[6] [SP 800-38F]	AES KW, KWP	Key sizes: 128, 192, 256 bits (Key establishment methodology provide 128, 192 or 256 bits of encryption strength)	Key Wrapping

5 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.

6 Keys are not established directly into the module using key unwrapping.

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Sizes(s)/Key Strength(s)	Use/Function
A4270	LMS [SP 800-208]	N/A	Parameters*: LMS-SHA256-M24-H10, LMS-SHA256-M24-H15, LMS-SHA256-M24-H20, LMS-SHA256-M24-H25, LMS-SHA256-M24-H5, LMS-SHA256-M32-H10, LMS-SHA256-M32-H15, LMS-SHA256-M32-H20, LMS-SHA256-M32-H25, LMS-SHA256-M32-H5, LMS-SHAKE-M24-H10, LMS-SHAKE-M24-H15, LMS-SHAKE-M24-H20, LMS-SHAKE-M24-H25, LMS-SHAKE-M24-H5, LMS-SHAKE-M32-H10, LMS-SHAKE-M32-H15, LMS-SHAKE-M32-H20, LMS-SHAKE-M32-H25, LMS-SHAKE-M32-H5 for W1, W2, W3, and W4. * keys only used for Signature Verification	Signature Verification
A4270	RSA [FIPS 186-5, ANSI X9.31-1998 and PKCS #1 v2.1 (PSS and PKCS1.5)]	N/A	Key sizes: 2048, 3073, 4096	Key Pair Generation
A4270	RSA [FIPS 186-5, FIPS 186-2, ANSI X9.31-1998 and PKCS #1 v2.1 (PSS and PKCS1.5)]	N/A	Key sizes: 2048, 3072, 4096	Signature Generation
A4270	RSA [FIPS 186-5, ANSI X9.31-1998 and PKCS #1 v2.1 (PSS and PKCS1.5)]	N/A	Key sizes: 2048, 3072, 4096	Signature Verification

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Sizes(s)/Key Strength(s)	Use/Function
A4270	KTS-IFC[7] [SP 800-56B-rev2, Section 7.2.2]	N/A	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	Key Transport
A4270	KAS-IFC[8] [SP 800-56B-rev2, Section 7.2.1]	N/A	RSASVE with, and without, key confirmation. Key sizes: 2048, 3072, 4096 providing between 112 and 152 bits of encryption strength	Key Agreement
A4270	Safe Primes [SP 800-56A-rev3]	N/A	Parameter sets: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Key Generation, Key Verification
A4270	SHS [FIPS 180-4]	N/A	SHA sizes: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256	Digital Signature Generation, Digital Signature Verification, non-Digital Signature Applications
A4270	SHA-3, SHAKE [FIPS 202]	N/A	SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256	Digital Signature Generation, Digital Signature Verification, non-Digital Signature Applications

7 KTS-IFC is an approved key transport method per FIPS 140-3 IG D.G
8 KAS-IFC is an approved key agreement method per FIPS 140-3 IG D.F.

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Sizes(s)/Key Strength(s)	Use/Function
A4270	SHA-3 Derived Functions [SP 800-185]	N/A	Types: cSHAKE-128, KMAC-128, TupleHash-128, ParallelHash-128, cSHAKE-256, KMAC-256, TupleHash-256, ParallelHash-256	Digital Signature Generation, Digital Signature Verification, non-Digital Signature Applications
Vendor Affirmed IG D.H	CKG using output from DRBG[9] [SP 800-133]	N/A	Section 4 (Asymmetric from DRBG) Section 4 (Symmetric from DRBG)	Key Generation

Table 4: Approved Algorithms

Algorithm	Caveat	Description
MD5 within TLS	[IG 2.4.A]	MD5 used within the TLS 1.0/1.1 handshake.

Table 5: Non-Approved Algorithms Allowed in the Approved Mode of Operation with No Security Claimed

9 The resulting key or a generated seed is an unmodified output from a DRBG

Algorithm/Function	Use/Function
AES (non-compliant[10])	Non-FIPS 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[11])	non-compliant key agreement methods
DSA (non-compliant[12])	non-FIPS digest signatures using DSA
DSTU4145	DSTU4145 EC algorithm
ECDSA (non-compliant[12])	non-FIPS digest signatures using ECDSA
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

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

Algorithm/Function	Use/Function
HSS	HSS signature scheme (RFC 8708)
IDEA	IDEA block cipher
KAS[13] using SHA-512/224 or SHA-512/256	Key Agreement using SHA-512/224 and SHA-512/256 based KDFs
PBKDF using SHA-512/224 or SHA-512/256 (non-compliant)	PBKDF2 using the PRFs SHA-512/224 and SHA-512/256
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
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

Table 6: Non-Approved Algorithms Not Allowed in the Approved Mode of Operation

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

2.1 Basic Enforcement

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 shall provide two distinct operator roles: User and Cryptographic Officer.
2. The module does not provide authentication.
3. The operator shall be capable of commanding the module to perform the power up self-tests by cycling power or resetting the module.
4. Power up self-tests do not require any operator action.
5. Data output shall be 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 tested operational environments physical perimeter.
11. The module does not output intermediate key values.

HMAC algorithms specified in the Approved Algorithms table produce truncated versions of the HMAC in question. The right most bits are truncated as per the NIST SP 800-107 rev1.

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 a non-approved mode by default.

From non-approved mode to approved mode:
It 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 indicating within the class (object), instantiating the key, as being created in an approved mode or non-approved mode. Any attempt by a thread within the execution of 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 see if the thread making the request is in the same mode.

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

2.2 Enforcement and Guidance for GCM IVs

IVs for GCM can be generated randomly, or via a FipsNonceGenerator. Where an IV is not generated within the module the module supports the importing of GCM IVs.

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.

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.

In approved mode, importing a GCM IV for encryption that originates from outside the module is non-conformant. 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, 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 TLSv1.2 protocol: The module is compatible with TLSv1.2 protocol and supports acceptable AES-GCM ciphersuites from Section 3.3.1 of the SP800-52rev2.

2.3 Enforcement and Guidance for use of the Approved PBKDF

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. Care should be taken where a password is simply based on ASCII. A 14 byte ASCII password is unlikely to contain sufficient entropy for most purposes as the standard set of printable characters only allows for as much as 6 bits of entropy per byte. In the case of a 14 byte password, this yields a key that has been generated using 14 * 6 bits, giving only 84 bits of security, 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 where the 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 the 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.

2.4 Rules for setting the N and the S String in cSHAKE

The cSHAKE algorithm offers to input string for customizing the output of the cSHAKE function, 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.

2.5 Guidance for the use of Format-Preserving Encryption

The module supports both FF1 and, in non-approved mode, FF3-1 format preserving encryption. Below shows the parameter constraints applicable to the module’s implementation.

SP800-38G Format-Preserving Encryption Constraints

	FF1	***FF3-1***
radix	in range of $2..2^{16}$	in range of $2..2^{16}$

$radix^{minlen}$	>= 1000000	>= 1000000
minlen	>= 2 octets	2 octets
maxlen	$< 2^{32}$ octets	2 * $floor(log_{radix}(2^{96}))$ octets
maxTlen	>= 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 Cryptographic Key Generation

The module performs Cryptographic Key Generation in conformance to FIPS 140-3 IG D.H. Symmetric keys and seeds used for asymmetric keys are generated per Section 4 of the SP800-133r2.

3 Cryptographic Module Ports and Interfaces

The BC-FJA (Bouncy Castle FIPS Java API) Module is a software module, and, therefore, control of the physical ports is outside of the module's scope. The module does provide a set of logical interfaces which are mapped to the following FIPS 140-3 defined logical interfaces: data input, data output, control input, status output, and power. 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. The module does not implement control output interface.

The mapping of the FIPS 140-3 logical interfaces to the module is described in Table 7.

Interface	Module Equivalent
Data Input	API input parameters – plaintext and/or ciphertext data.
Data Output	API output parameters and return values – plaintext and/or ciphertext data.
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.
Status Output	API output parameters and return/error codes that provide status information used to indicate the state of the module.

Table 7: Ports and Interfaces

4 Roles, Services, and Authentication

4.1 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 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 being used in approved mode, classes providing implementations of algorithms which 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.

Roles, with corresponding service with input and output is specified in Table 8 below:

Role	Service	Input	Output
CO/User	Initialize Module and Run Self-Tests on Demand	N/A	Exception in case of failure
CO/User	Show Status	N/A	Boolean
CO/User	Info Service	N/A	Module name and version
CO/User	Zeroize / Power-off	N/A	Shutdown indication
CO/User	Data Encryption	Key, Plaintext	Ciphertext
CO/User	Data Decryption	Key, Ciphertext	Plaintext
CO/User	MAC Calculation	Key, Message	MAC
CO/User	Signature Authentication	Key, Message	Signature
CO/User	Signature Verification	Key, Message, Signature	Boolean
CO/User	DRBG (SP800-90Arev1) Output	N/A	Data
CO/User	Message Hashing	Message	Hash
CO/User	Keyed Message Hashing	Key, Message	Hash
CO/User	TLS Key Derivation Function	TLS Parameters	Key

Role	Service	Input	Output
CO/User	SP 800-108-rev1 KDF	KDF Parameters	Key
CO/User	SSH Derivation Function	SSH Parameters	Key
CO/User	X9.63 Derivation Function	X9.63 Parameters	Key
CO/User	SP 800-56C-rev2 OneStep/TwoStep Key Derivation Function (KDM)	KDM Parameters	Key
CO/User	IKEv2 Derivation Function	IKEv2 Parameters	Key
CO/User	SRTP Derivation Function	SRTP Parameters	Key
CO/User	PBKDF	Password, PBKDF Parameters	Key
CO/User	Key Agreement Schemes	Key Agreement keys, parameters	Shared Secret
CO/User	Key Wrapping	Wrapping key, Key	Wrapped key
CO/User	Key Unwrapping	Unwrapping Key, Wrapped key	Key
CO/User	Key Generation	Key Generation Parameters	Keys or Key Pairs
CO/User	Key Verification	Key Pair	Boolean
CO/User	Entropy Callback	N/A	Random bits
CO/User	SSP Export Operation	Command	SSP
CO/User	Utility	N/A	N/A

Table 8: Roles, Service Commands, Input and Output

4.2 Assumption of Roles

The module supports two distinct operator roles, User and Cryptographic Officer (CO). The cryptographic module implicitly maps the two roles to the services. A user is considered the owner of the thread that instantiates the module and, therefore, only one concurrent user is allowed.
Table 9 lists all operator roles supported by the module. The module does not support a maintenance role and/or bypass capability. The module does not support authentication.

Role ID	Authentication Method	Authentication Strength
CO	N/A – Authentication not required for Level 1	N/A
User	N/A – Authentication not required for Level 1	N/A

Table 9: Roles and Authentication

4.3 Services

Table 10 lists the services and a description of each service with the usage and roles.

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 only mode no non-approved services are accessible. 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 permissions to be configured in the JVM configuration by the Cryptographic Officer or User.

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.

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator[16]
Initialize Module and Run Self-Tests on Demand	The JRE will call the static constructor for self-tests on module initialization.	HMAC-SHA-256	N/A	CO/User	N/A	Flag
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 FIPS mode of operation.	N/A	N/A	CO/User	N/A	Flag
Info Service	A user can call DumpInfo.main() at any time to display the module version, checksum, and status information.	N/A	N/A	CO/User	N/A	Flag
Zeroize / Power-off	The module uses the JVM garbage collector on thread termination.	N/A	All SSPs	CO/User	Z	Flag
Data Encryption	Used to encrypt data.	AES-ECB, AES-CBC, AES-OFB, AES-CFB8, AES-CFB128, AES-CTR, AES-CBC-CS, CCM, GCM, FF1	AES Encryption Key	CO/User	W/E	Flag

16 Flag is accessed by calling the method CryptoServicesRegistrar.isInApprovedOnlyMode() - this method will return true if the thread is running in approved mode, false otherwise.

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Data Decryption	Used to decrypt data.	AES-ECB, AES-CBC, AES-OFB, AES-CFB8, AES-CFB128, AES-CTR, AES-CBC-CS, CCM, GCM, FF1	AES Decryption Key	CO/User	W/E	Flag
MAC Calculation	Used to calculate data integrity codes with CMAC.	CMAC, GMAC	AES Authentication Key	CO/User	W/E	Flag
Signature Authentication	Used to generate signatures (DSA, ECDSA, EdDSA, RSA).	DSA, ECDSA, EdDSA, RSA	DSA Signing Key, EC Signing Key, EdDSA Signing Key, RSA Signing Key	CO/User	W/E	Flag
Signature Verification	Used to verify digital signatures.	DSA, ECDSA, EdDSA, LMS, RSA	DSA Verification Key, EC Verification Key, EdDSA Verification Key, LMS Verification Key, RSA Verification Key	CO/User	W/E	Flag
DRBG (SP800-90A) output	Used for random number, IV and key generation.	Counter DRBG, Hash DRBG, HMAC DRBG	Entropy Input String, DRBG Seed, Internal State V and C value, and DRBG Key	CO/User	W/G/E	Flag
Message Hashing	Used to generate message digest, SHAKE output	SHS, SHA-3, SHAKE, SHA-3 Derived Functions (cSHAKE, TupleHash, ParallelHash)	N/A	CO/User	N/A	Flag

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Keyed Message Hashing	Used to calculate data integrity codes with HMAC and KMAC.	HMAC, SHA-3 Derived Functions (KMAC)	HMAC Authentication Key, KMAC Authentication Key	CO/User	W/E	Flag
TLS Key Derivation Function	Used to calculate a value suitable to be used for a master secret in TLS.	HKDF, KDF, Existing Application-Specific (TLS KDF)	TLS Premaster secret TLS KDF Secret Value	CO/User	W/E/R	Flag
SP 800-108-rev1 KDF	Used to calculate a value suitable to be used for a secret key	KBKDF, using Pseudorandom Functions	SP800-108-rev1 KDF Secret Value	CO/User	W/E/R	Flag
SSH Derivation Function	Used to calculate a value suitable to be used for a secret key	Existing Application-Specific (SSH KDF)	SSH KDF Secret Value	CO/User	W/E/R	Flag
X9.63 Derivation Function	Used to calculate a value suitable to be used for a secret key	Existing Application-Specific (X9.63 KDF)	X9.63 KDF Secret Value	CO/User	W/E/R	Flag
SP 800-56C OneStep/TwoStep Key Derivation Function (KDM)	Used to calculate a value suitable to be used for a secret key	HKDF, KDF One Step, KDF Two Step.	SP800-56C-rev2 KDF Secret Value	CO/User	W/E/R	Flag
IKEv2 Derivation Function	Used to calculate a value suitable to be used for a secret key	Existing Application-Specific (IKEv2 KDF)	IKEv2 KDF Secret Value	CO/User	W/E/R	Flag
SRTP Derivation Function	Used to calculate a value suitable to be used for a secret key	Existing Application-Specific (SRTP KDF)	SRTP KDF Secret Value	CO/User	W/E/R	Flag

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
PBKDF	Used to generate a key using an encoding of a password and message hash	KDF, Password-Based	HMAC Authentication Key, KMAC Authentication Key	CO/User	G/R	Flag
			PBKDF Secret	CO/User	W/E	
Key Agreement Schemes	Used to calculate key agreement values (SP 800-56A-rev3, Diffie-Hellman).	KAS-FFC, KAS-ECC, KAS-IFC, SafePrimes	DH Agreement Private Key, DH Agreement Public Key, EC Agreement Private Key, EC Agreement Public Key, RSA Key Transport Private Key, RSA Key Transport Public Key.	CO/User	W/E/R	Flag
Key Wrapping	Used to encrypt a key value. (RSA, AES)	AES KW, AES KWP, KTS-IFC	AES Wrapping Key, HMAC Authentication Key, KMAC Authentication Key, RSA Key Transport Public Key	CO/User	W/E	Flag
Key Unwrapping	Used to decrypt a key value. (RSA, AES)	AES KW, AES KWP, KTS-IFC	AES Wrapping Key, HMAC Authentication Key, KMAC Authentication Key, RSA Key Transport Private Key	CO/User	W/E	Flag

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Key Generation	Used to generate symmetric keys and asymmetric key pairs	RSA KeyGen, DSA KeyGen, ECDSA KeyGen, EdDSA KeyGen, SafePrimes KeyGen, CKG	DRBG Output, DSA Signing Key, EC Signing Key, EdDSA Signing Key, RSA Signing Key, DSA Verification Key, EC Verification Key, EdDSA Verification Key, RSA Verification Key, DH Agreement Private Key, DH Agreement Public Key, EC Agreement Private Key, EC Agreement Public Key, RSA Key Transport Private Key, RSA Key Transport Public Key, AES Encryption Key, AES Decryption Key, HMAC Authentication Key, KMAC Authentication Key	CO/User	W/G/R	Flag
Key Verification	Used to verify key pair	ECDSA KeyVer, EdDSA KeyVer	EC Signing Key, EC Verification Key, EC Agreement Public Key, EC Agreement Private Key, EdDSA Signing Key, EdDSA Verification Key	CO/User	W/E	Flag
Entropy Callback	Gathers entropy in a passive manner from a user-provided function	DRBG, CKG	Entropy input string	CO/User	W	Flag

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
SSP Export Operation	Returns a CSP as data that can be used for later output	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, EdDSA Signing Key, EdDSA Verification Key, HMAC Authentication Key, KMAC Authentication Key, LMS Verification Key, RSA Signing Key, RSA Verification Key, RSA Key Transport Private Key, RSA Key Transport Public Key	CO/User	R	Flag
Utility	Miscellaneous utility functions, does not access CSPs	N/A	N/A	CO/User	N/A	Flag

Table 10: Approved Services

The modes of access shown in the table above 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.

Service	Description	Algorithms Accessed	Role	Indicator[17]
Data Encryption	Used to encrypt data	Triple-DES	CO/User	Flag
Data Decryption	Used to decrypt data	Triple-DES	CO/User	Fla
MAC Calculation	Used to calculate data integrity codes with CMAC	Triple-DES CMAC	CO/User	Flag
DRBG (SP800-90Arev1) output	Used for random number, IV and key generation	ctrDRBG-Triple-DES	CO/User	Flag
Key Agreement Schemes	Used to calculate key agreement values	Triple-DES	CO/User	Flag
Key Wrapping	Used to encrypt a key value (Triple-DES)	Triple-DES KW	CO/User	Flag
Key Unwrapping	Used to decrypt a key value (Triple-DES)	Triple-DES KW	CO/User	Flag

Table 11: Non-Approved Services

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.

5 Software/Firmware Security

The Module type is software. The module has a Multi-Chip Stand Alone embodiment; the cryptographic boundary is the Java Archive (JAR) file, *bc-fips-2.1.0.jar*.

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.

The integrity is verified using HMAC-SHA2-256. Using the a 256-bit embedded key in the module jar, 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.

CASTs are preformed prior to the first use of services related to the test target. CASTs also run periodically on service invocation. Initial CAST self–tests are available on demand by power cycling the module and then invoking the service related to the test target.

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 in Table 2. Each approved operating system manages processes and threads in a logically separated manner. The Module's user is considered the owner of the calling application that instantiates the Module within the process space of the Java Virtual Machine. The module optionally uses the Java Security Manager and starts in approved mode by default when used with the Java Security Manager.

6.1 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. The module supports the definition of user-defined DRBGs using the definitions available in SP 800-90A. In approved mode the minimum amount of entropy that can be requested by a user defined DRBG is 112 bits. The module will wait until the *SecureRandom.generateSeed()* returns the requested amount of entropy, blocking if necessary.

The JVMs entropy source can be configured through setting the security property:

securerandom.strongAlgorithms

in the JVM's java.security file.

6.2 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. See Section 6.3 for further advice.

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.

6.3 Approved Mode Configuration

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.

If the underlying JVM is running with a Java Security Manager installed the module will be running in approved mode with secret and private key export disabled. When the module is not used within the context of the Java Security Manager, it will start by default in the non-approved mode.

Use of the module with a Java Security manager 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 Security manager are indicated by a Y:

Available Java Permissions

Permission	Settings	Req	Usage
RuntimePermission	“getProtectionDomain”	Y	Allows checksum to be carried out on jar.
RuntimePermission	“accessDeclaredMembers”	Y	Allows use of reflection API within the provider.
PropertyPermission	“java.runtime.name”, “read”	N	Only if configuration properties are used.
SecurityPermission	"putProviderProperty.BCFIPS"	N	Only if provider installed during execution.
CryptoServicesPermission	“unapprovedModeEnabled”	N	Only if non-approved mode algorithms required.
CryptoServicesPermission	“changeToApprovedModeEnabled”	N	Only if threads allowed to change modes.
CryptoServicesPermission	“exportSecretKey”	N	To allow export of secret keys only.
CryptoServicesPermission	“exportPrivateKey”	N	To allow export of private keys only.
CryptoServicesPermission	“exportKeys”	Y	Required to be applied for the module itself. Optional for any other codebase.
CryptoServicesPermission	“tlsNullDigestEnabled”	N	Only required for TLS digest calculations.
CryptoServicesPermission	“tlsPKCS15KeyWrapEnabled”	N	Only required if TLS is used with RSA encryption.
CryptoServicesPermission	“tlsAlgorithmsEnabled”	N	Enables both NullDigest and PKCS15KeyWrap.
CryptoServicesPermission	“defaultRandomConfig”	N	Allows setting of default SecureRandom.
CryptoServicesPermission	“threadLocalConfig”	N	Required to set a thread local property in the CryptoServicesRegistrar
CryptoServicesPermission	“globalConfig”	N	Required to set a global property in the CryptoServicesRegistrar.

6.4 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 strength 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 suggest C values for the Repetition Count Test (Section 4.4.1) and the Adaptive Proportion Test (Section 4.4.2). These values can also be configured by the user using the security property: "*org.bouncycastle.entropy.factors*" which takes a comma separated list of C values, one for 4.4.1 and 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 6.1 of this Security Policy and will block, or fail, if it is unable to provide the amount of entropy requested.

7 Physical Security

This section is not applicable as the module is a software only module.

8 Non-invasive Security

This section is not applicable to this module.

9 Sensitive Security Parameter Management

All Sensitive Security Parameters (SSPs) used by the Module are described in this section in Table 12. All usage of these SSPs by the Module (including all SSP lifecycle states) is described in the services detailed in Section 4.3. 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.

Key/SSP Name/ Type	Strength	Security Function and Cert. Number	Generation	Import/ Export	Establishment	Storage	Zeroisation	Use & related keys
AES Encryption Key	128, 192, 256 bits	AES ECB, CBC, OFB, CFB8, CFB128, CTR, FF1, CBC-CS1, CBC-CS2, CBC-CS3, GCM A4270	KDF, CKG[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	AES encryption[21]
AES Decryption Key	128, 192, 256 bits	AES ECB, CBC, OFB, CFB8, CFB128, CTR, FF1, CBC-CS1, CBC-CS2, CBC-CS3, GCM A4270	KDF,CKG[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	AES decryption
AES Authentication Key	128, 192, 256 bits	AES CMAC, GMAC A4270	KDF,CKG[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	AES CMAC/GMAC

18 Key generator used in conjunction with an approved DRBG.
19 Import done via key constructor
20 Export done via accessing returned key object using getEncoded() method and followed by separate step to export key details as either plaintext or encrypted
21 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.

Key/SSP Name/ Type	Strength	Security Function and Cert. Number	Generation	Import/ Export	Establishment	Storage	Zeroisation	Use & related keys
AES Wrapping Key	128, 192, 256 bits	AES KW, KWP A4270	KDF, CKG[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Key wrapping key
DH Agreement Private Key	112, 128, 152, 176, 200 bits	KAS-FFC A4270	DSA KeyGen, SafePrimes KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Diffie-Hellman key agreement
DH Agreement Public Key	112, 128, 152, 176, 200 bits	KAS-FFC A4270	DSA KeyGen, SafePrimes KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	Not zeroized, public key value known outside of modulePrivate	Diffie-Hellman key agreement
DSA Signing Key	112, 128 bits	DSA Signature Generation A4270	DSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	DSA signature generation
DSA Verification Key	80, 112, 128 bits	DSA Signature Verification A4270	DSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	Not zeroized, public key value known outside of module	DSA signature verification
EC Agreement Private Key	112. 128. 192. 256 bits	KAS-ECC A4270	ECDSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	EC key agreement
EC Agreement Public Key	112. 128. 192. 256 bits	KAS-ECC A4270	ECDSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	Not zeroized, public key value known outside of module	EC key agreement

Key/SSP Name/ Type	Strength	Security Function and Cert. Number	Generation	Import/ Export	Establishment	Storage	Zeroisation	Use & related keys
EC Signing Key	112, 128, 192, 256 bits	ECDSA Signature Generation A4270	ECDSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	ECDSA signature generation.
EC Verfication Key	80, 112, 128, 192, 256 bits	ECDSA Signature Verification A4270	ECDSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	Not zeroized, public key value known outside of module	ECDSA signature verification.
EdDSA Signing Key	128, 224 bits	EdDSA Signature Generation A4270	EdDSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	EdDSA signature generation.
EdDSA Verfication Key	128, 224 bits	EdDSA Signature Verification A4270	EdDSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	Not zeroized, public key value known outside of module	EdDSA signature verification.
HMAC/KMAC Authentication Key	112-256 bits	SHA-1, SHA2, SHA3, KMAC A4270	KDF,CKG[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Keyed-Hash calculation.
LMS Verfication Key	128, 192, 256 bits	LMS Signature Verification A4270	LMS KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	Not zeroized, public key value known outside of module	LMS signature verification.
RSA Signing Key	112, 128, 152 bits	RSA Signature Generation A4270	RSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	RSA signature generation

Key/SSP Name/ Type	Strength	Security Function and Cert. Number	Generation	Import/ Export	Establishment	Storage	Zeroisation	Use & related keys
RSA Verification Key	80, 112, 128, 152 bits	RSA Signature Verfication A4270	RSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	Not zeroized, public key value known outside of module	RSA signature verification
RSA Key Transport Private Key[22]	112, 128, 152 bits	KTS-IFC A4270	RSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	RSA key transport and decryption
RSA Key Transport Public Key[22]	112, 128, 152 bits	KTS-IFC A4270	RSA KeyGen[18]	Import[19], Export[20]	N/A	SDRAM in plaintex	Not zeroized, public key value known outside of module	RSA key transport
IKEv2 KDF Secret Value	128, 192 bits	KDF IKEv2 A4270	Generated as output of an IKEv2 agreement scheme	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Key Derivation
PBKDF Secret Value	192 bits	PBKDF A4270	Generated as output of a PBE key and a PRF	Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Key Derivation
SP 800-56C-rev2 OneStep/TwoStep KDF Secret Value	112, 128, 192, 256 bits	KDA OneStep SP800-56Cr2, KDA TwoStep SP800-56Cr2 A4270	Generated as output of an agreement scheme	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Key Derivation

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

Key/SSP Name/ Type	Strength	Security Function and Cert. Number	Generation	Import/ Export	Establishment	Storage	Zeroisation	Use & related keys
SP 800-108-rev1 KDF Secret Value	112, 128, 192, 256 bits	KDF SP800-108 A4270	Generated as output of an agreement scheme	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Key Derivation
SRTP KDF Secret Value	128, 192, 256 bits	KDF SRTP A4270	Generated as output of an SRTP agreement scheme	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Key Derivation
SSH KDF Secret Value	80, 112, 128, 192, 256 bits	KDF SSH A4270	Generated as output of an SSH agreement scheme	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Key Derivation
TLS Premaster Secret Value	384 bits	KDF TLS A4270	Protocol version (2 bytes) and 46 bytes from a DRBG[18]	Import[19]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Key Derivation
TLS KDF Secret Value	112, 128, 192, 256 bits	KDF TLS A4270	Generated as output of a TLS agreement scheme	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Key Derivation
X9.63 KDF Secret Value	112, 128, 192, 256 bits	KDF ANS 9.63 A4270	Generated as output of an agreement scheme	Import[19], Export[20]	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Key Derivation

Key/SSP Name/ Type	Strength	Security Function and Cert. Number	Generation	Import/ Export	Establishment	Storage	Zeroisation	Use & related keys
Entropy Input String	>128 bits	N/A	N/A	Obtained from the entropy source	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Random Number Generation
CTR DRBG Seed	128, 192, 256 bits	N/A	CTR DRBG	N/A	N/A	SDRAM in plaintex	Immediately after use or host platform power cycle	Random Number Generation
CTR DRBG V Value	128 bits	N/A	CTR DRBG	N/A	N/A	SDRAM in plaintex	reseed() service call or host platform power cycle	Random Number Generation
CTR DRBG Key	128, 192, 256 bits	N/A	CTR DRBG	N/A	N/A	SDRAM in plaintex	reseed() service call or host platform power cycle	Random Number Generation
Hash DRBG Seed	112, 128, 192, 256 bits	N/A	Hash DRBG	N/A	N/A	SDRAM in plaintex	Immediately after use or host platform power cycle	Random Number Generation
Hash DRBG V Value	112, 128, 192, 256 bits	N/A	Hash DRBG	N/A	N/A	SDRAM in plaintex	reseed() service call or host platform power cycle	Random Number Generation
Hash DRBG C Value	112, 128, 192, 256 bits	N/A	Hash DRBG	N/A	N/A	SDRAM in plaintex	reseed() service call or host platform power cycle	Random Number Generation
HMAC DRBG Seed	112, 128, 192, 256 bits	N/A	HMAC DRBG	N/A	N/A	SDRAM in plaintex	Immediately after use or host platform power cycle	Random Number Generation

Key/SSP Name/ Type	Strength	Security Function and Cert. Number	Generation	Import/ Export	Establishment	Storage	Zeroisation	Use & related keys
HMAC DRBG V Value	112, 128, 192, 256 bits	N/A	HMAC DRBG	N/A	N/A	SDRAM in plaintex	reseed() service call or host platform power cycle	Random Number Generation
HMAC DRBG Key	112, 128, 192, 256 bits	N/A	HMAC DRBG	N/A	N/A	SDRAM in plaintex	reseed() service call or host platform power cycle reseed() service call or host platform power cycle	Random Number Generation
DRBG Output	512 to 2048 bits	N/A	DRBG	N/A	N/A	SDRAM in plaintex	destroy() service call or host platform power cycle	Used as seed for asymmetric key generation or for symmetric key generation

Table 12: SSPs

9.1 RBG Entropy Sources

The module's use of Non-Deterministic Random Number Generators is determined by the settings described in Section 6.1.

Entropy Sources	Minimum number of bits of entropy	Description / Usage
Passive Entropy	128	A minimum of 16 bytes 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.

Table 13: Non-Deterministic Random Number Generation Specification

The module passively receives entropy from a source within the physical perimeter of the tested operational environment, in conformance with FIPS 140-3 IG 9.3.A scenario 1b. The following entropy caveat applies: “The module generates SSPs (e.g., keys) whose strengths are modified by available entropy.”

10 Self-Tests

CASTs are performed prior to the first use of services related to the test target. CASTs also run periodically on service invocation. Initial CAST self–tests are available on demand by power cycling the module and then invoking the service related to the test target.

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 have not been damaged. The pre-operational tests include the Software Integrity test, which verifies the module using HMAC-SHA2-256. The HMAC and SHS Conditional Cryptographic Algorithm Self-Tests (CAST) 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. Below are the self-tests implemented:

Conditional Cryptographic Algorithm Self-Test:

- 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)
- KAS-ECC Primitive “Z” Computation KAT (P-256, B-233)
- KAS-FFC Primitive “Z” Computation KAT (ffdhe2048)
- HASH-DRBG SHA2-256 KAT (Health Tests: Generate, Reseed, Instantiate functions per Section 11.3 of SP800-90Arev1)
- HMAC-DRBG HMAC-SHA2-256 KAT (Health Tests: Generate, Reseed, Instantiate functions per Section 11.3 of SP800-90Arev1)
- CTR-DRBG AES-CTR 256 bits KAT (Health Tests: Generate, Reseed, Instantiate functions per Section 11.3 of SP800-90Arev1)
- DSA Signature Generation KAT (2048 bits)
- DSA Signature Verification KAT (2048 bits)
- ECDSA Signature Generation KAT (P-256)
- ECDSA Signature Verification KAT (P-256)
- EdDSA Signature Generation KAT (Ed25519, Ed448)
- EdDSA Signature Verification KAT (Ed25519, Ed448)
- EdDSA Signature Generation (prehash) KAT (Ed25519, Ed448)
- EdDSA Signature Verification (prehash) KAT (Ed25519, Ed448)
- AES-GCM Encrypt KAT (128 bits)
- AES-GCM Decrypt KAT (128 bits)
- HMAC-SHA2-256 KAT
- HMAC-SHA2-512 KAT
- HMAC-SHA3-256 KAT
- KDA OneStep KAT
- KDA TwoStep KAT

- KBKDF KAT (Counter, Feedback, Double Pipeline)
- LMS Signature Verification KAT (Parameters: LMS-SHA256-N24-H5/W2)
- PBKDF KAT (HMAC-SHA2-256)
- SHA-3 KAT (cSHAKE-128)
- RSA Signature Generation KAT (2048 bits)
- RSA Signature Verification KAT (2048 bits)
- RSA Encryption KAT SP800-56Brev2 (2048 bits)
- RSA Decryption KAT SP800-56Brev2 (2048 bits)
- SHA-1 KAT
- SHA2-256 KAT
- SHA2-512 KAT
- SHAKE256 KAT
- ANS 9.63 KDF KAT
- IKEv2 KDF KAT
- SNMP KDF KAT
- SRTP KDF KAT
- SSH KDF KAT
- TLS 1.0/1.1 KDF KAT
- TLS 1.2 KDF KAT
- TLS 1.3 KDF KAT

Conditional Pair-wise Consistency Tests:
- DH pair-wise consistency test
- DSA pair-wise consistency test
- EC DH pair-wise consistency test
- ECDSA pair-wise consistency test
- RSA pair-wise consistency test

10.3 Error Handling

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 mentioned below:

> “Failed self-test on encryption: AES”

The module can be recovered by power cycling the module 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.

11 Life-cycle Assurance

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

Researchers and users are encouraged to report any security related concerns to feedback-crypto@bouncycastle.org. A PGP public key can be provided if confidentiality is required around the report.

Please find the procedures for secure installation, initialization, startup and operation of the module:
The module exists as part of the running JVM as such:

- Secure installation of the module requires the use of the unchanged jar to 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 can be 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 doing any required CASTs and health tests and then provide a service object to the user capable of performing the requested service.

BC-FJA 2.1.0 User guide can be downloaded here:
https://downloads.bouncycastle.org/fips-java/BC-FJA-UserGuide-2.1.0.pdf

12 Mitigation of Other Attacks

The Module implements basic protections to mitigate against timing based attacks against its internal implementations. There are two counter-measures used.

The first 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 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 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 Definitions

The following standards are referred to in this Security Policy.

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 186-3	*Digital Signature Standard (DSS)*
FIPS 186-5	*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 StandardRecommendation 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 800-56A	*Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*
SP 800-56B	*Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography*
SP 800-56C	*Recommendation for Key Derivation through Extraction-then-Expansion*
SP 800-67	*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*
SP 800-108	*Recommendation for Key Derivation Using Pseudorandom Functions*
SP 800-132	*Recommendation for Password-Based Key Derivation*
SP 800-133	*Recommendation for Cryptographic Key Generation*

ANSI X9.31	*X9.31-1998, Digital Signatures using Reversible Public Key Cryptography for the Financial Services Industry (rDSA), September 9, 1998*
SP 800-135	*Recommendation for Existing Application – Specific Key Derivation Functions*

The following are acronyms used in this Security Policy:

AES	Advanced Encryption Standard
API	Application Programming Interface
BC	Bouncy Castle
BC-FJA	Bouncy Castle FIPS Java API
CBC	Cipher-Block Chaining
CCM	Counter with CBC-MAC
CDH	Computational Diffie-Hellman
CFB	Cipher Feedback Mode
CMAC	Cipher-based Message Authentication Code
CMVP	Crypto 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 Authority
DSTU4145	Ukrainian DSTU-4145-2002 Elliptic Curve Scheme
EC	Elliptic Curve
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Authority
EdDSA	Edwards Curve DSA using Ed25519, Ed448
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FIPS	Federal Information Processing Standards
GCM	Galois/Counter Mode
GMAC	Galois Message Authentication Code
GOST	Gosudarstvennyi Standard Soyuza SSR/Government Standard of the Union of Soviet Socialist Republics

AES	Advanced Encryption Standard
GPC	General Purpose Computer
HMAC	key-Hashed Message Authentication Code
IG	See References
JAR	Java ARchive
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
IV	Initialization Vector
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	Non Applicable
NDRNG	Non Deterministic Random Number Generator
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
TCBC	TDEA Cipher-Block Chaining
TCFB	TDEA Cipher Feedback Mode
TDEA	Triple Data Encryption Algorithm
TDES	Triple Data Encryption Standard
TECB	TDEA Electronic Codebook
TOFB	TDEA Output Feedback
TLS	Transport Layer Security
USB	Universal Serial Bus
XDH	Edwards Curve Diffie-Hellman using X25519, X448
XOF	Extendable-Output Function