Dell Australia Pty Limited, BSAFE Product Team

Dell BSAFE Crypto Module for Java

FIPS 140-3 Non-Proprietary Security Policy

Table of Contents

List of Tables

List of Figures

Preface

With the exception of this non-proprietary *Dell BSAFE Crypto Module for Java Security Policy*, the FIPS 140-3 Security Level 1 validation submission documentation is proprietary to Dell Inc. and is releasable only under appropriate non-disclosure agreements. For access to the documentation, please contact Dell Support.

This security policy describes how the BSAFE Crypto Module meets the overall Security Level 1 security requirements of FIPS 140-3, and how to securely operate it.

Federal Information Processing Standards Publication 140-3 - Security Requirements for Cryptographic Modules (FIPS 140-3) details the U.S. Government requirements for cryptographic modules. More information about the FIPS 140-3 standard and validation program is available on the NIST website.

This document deals only with operations and capabilities of the BSAFE Crypto Module in the technical terms of a FIPS 140-3 cryptographic module security policy. More information about the BSAFE Crypto Module and the entire BSAFE product line is available at Dell Support.

Terminology

In this document, the term *BSAFE Crypto Module* denotes the BSAFE Crypto Module version 7.1, FIPS 140-3 validated Cryptographic Module for Overall Security Level 1 for the Java language.

The BSAFE Crypto Module is also referred to as:

- The Cryptographic Module
- The JCM
- The module.

1 General

This document is a non-proprietary security policy for the Dell BSAFE Crypto Module for Java 7.1 (BSAFE Crypto Module) security software from Dell Australia Pty Limited, BSAFE Product Team.

1.1 Overview

The JCM is validated for FIPS 140-3 Security Level 1 requirements. Security levels for individual areas are shown in the following table:

1.2 Security Levels

The BSAFE Crypto Module is validated with an overall Security Level 1 for FIPS 140-3, with Security Level 1 for each individual area.

Section	Title	Security Level
1	General	1
2	Cryptographic module specification	1
3	Cryptographic module 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
	Overall Level	1

Table 1: Security Levels

2 Cryptographic Module Specification

2.1 Description

2.1.1 Purpose and Use:

BSAFE Crypto Module is a software module intended to be used as part of a software system, providing cryptographic services to that system.

The module is provided as a Java Archive (jar) file. It is intended to be distributed with, and used by, a Java application.

Module Type: Software
Module Embodiment: Multi-Chip Standalone
Module Characteristics:

2.1.2 Cryptographic Boundary:

BSAFE Crypto Module is classified as a multi-chip standalone software cryptographic module for the purposes of FIPS 140-3. As such, it is tested on specific operating systems and computer platforms. The cryptographic boundary includes the module running on selected platforms running selected operating systems. The module is packaged as a jar file (jcmFIPS-7.1.jar) containing the module's entire executable code. The module relies on the physical security provided by the host computer in which it runs.

2.1.3 Tested Operational Environment's Physical Perimeter (TOEPP):

The physical perimeter of the module is the case of the general-purpose computer, which encloses the hardware running the module. The physical interfaces for the module are the physical interfaces of the computer running the module, such as the keyboard, monitor and network interface.

The following diagram illustrates the module's logical interfaces, physical perimeter and cryptographic boundary:

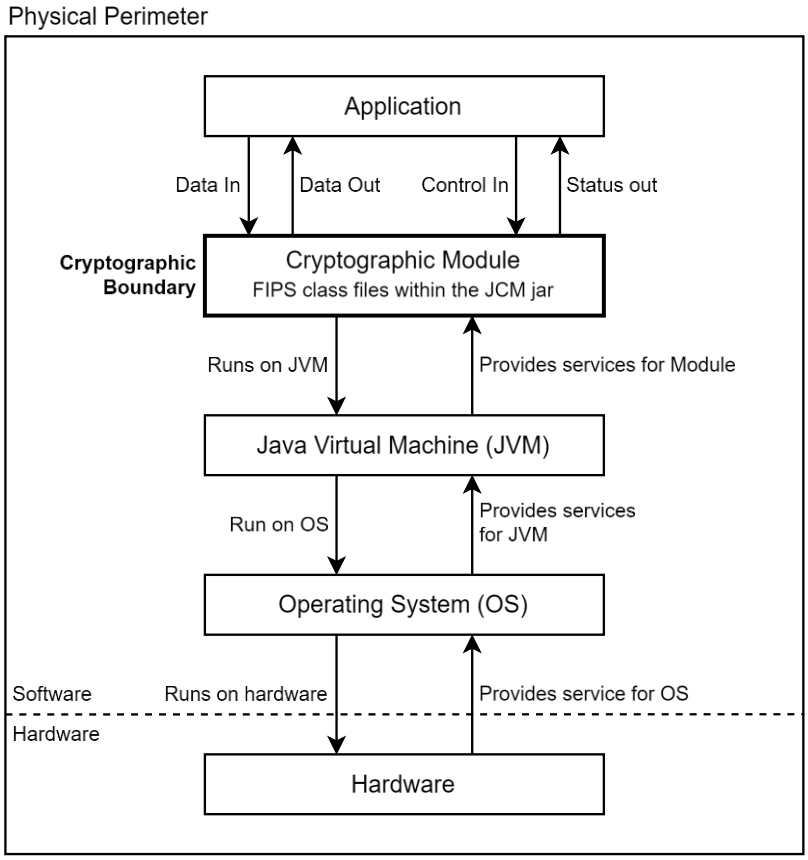

Figure 1: Block Diagram

2.2 Tested and Vendor Affirmed Module Version and Identification

For FIPS 140-3 validation, the module was tested by an accredited FIPS 140-3 testing laboratory on the following operational environments:

2.2.1 Tested Module Identification (Executable Code Sets):

Package or File Name	Software/ Firmware Version	Features	Integrity Test
jcmFIPS-7.1.jar	7.1		HMAC-SHA-1

Table 2: Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets)

2.2.2 Tested Operational Environments:

Operating System	Hardware Platform	Processors	PAA/PAI	Hypervisor or Host OS	Version(s)
Dell PowerProtect Data Domain OS 8.3 with Oracle JRE 8	Dell PowerEdge R760	Intel Xeon Silver 4510	No	ESXi 8.0.3	7.1
PowerStoreOS 4.1	Dell PowerStore 1200T	Intel Xeon Silver 4210R	No		7.1
SUSE Linux Enterprise Server 15 SP4 with OpenJDK 11	Dell PowerEdge R760	Intel Xeon Silver 4510	No	ESXi 8.0.3	7.1
SUSE Linux Enterprise Server 15 SP4 with OpenJDK 17	Dell PowerEdge R760	Intel Xeon Silver 4510	No	ESXi 8.0.3	7.1
SUSE Linux Enterprise Server 15 SP6 with OpenJDK 11	Dell PowerEdge R760	Intel Xeon Silver 4510	No	ESXi 8.0.3	7.1

Operating System	Hardware Platform	Processors	PAA/PAI	Hypervisor or Host OS	Version(s)
SUSE Linux Enterprise Server 15 SP6 with OpenJDK 17	Dell PowerEdge R760	Intel Xeon Silver 4510	No	ESXi 8.0.3	7.1
SUSE Linux Enterprise Server 15 SP6 with OpenJDK 21	Dell PowerEdge R760	Intel Xeon Silver 4510	No	ESXi 8.0.3	7.1
SUSE Linux Enterprise Server 15 SP6 with OpenJDK 8	Dell PowerEdge R760	Intel Xeon Silver 4510	No	ESXi 8.0.3	7.1
Windows Server 2019 with Oracle JRE 8	Dell PowerEdge R760	Intel Xeon Silver 4510	No	ESXi 8.0.3	7.1
Windows Server 2022 with OpenJDK 11	Dell PowerEdge R760	Intel Xeon Silver 4510	No	ESXi 8.0.3	7.1
Windows Server 2022 with OpenJDK 17	Dell PowerEdge R760	Intel Xeon Silver 4510	No	ESXi 8.0.3	7.1

Table 3: Tested Operational Environments - Software, Firmware, Hybrid

2.2.3 Vendor-Affirmed Operational Environments:

Dell BSAFE affirms compliance for the following operational environments:

Operating System	Hardware Platform
Dell PowerProtect Data Domain OS 7.13 (x64_64) with Oracle JRE 8	Intel x86_64
Dell PowerProtect Data Domain OS 8.3 with Oracle JRE 8	Dell PowerEdge R640
Dell PowerStoreOS 4 (x86_64)	Intel x86_64
Microsoft Windows 10 Enterprise (x86_64) with Oracle JRE 11	Intel x86_64
Microsoft Windows 10 Enterprise (x86_64) with Oracle JRE 8	Intel x86_64
Microsoft Windows Server 2016 (x86_64) with Oracle JRE 11	Intel x86_64
Microsoft Windows Server 2016 (x86_64) with Oracle JRE 8	Intel x86_64
Microsoft Windows Server 2019 (x86_64) with Oracle JRE 11	Intel x86_64
Microsoft Windows Server 2022 (x86_64) with OpenJDK 21	Intel x86_64
Red Hat Enterprise Linux 8.9 (x86_64) with Oracle JRE 11	Intel x86_64
Red Hat Enterprise Linux 8.9 (x86_64) with Oracle JRE 8	Intel x86_64
SUSE Linux Enterprise Server 12 SP5 (x86_64) with OpenJDK 8	Intel x86_64
SUSE Linux Enterprise Server 12 SP5 (x86_64) with Oracle JRE 11	Intel x86_64
SUSE Linux Enterprise Server 15 SP2 (x86_64) with OpenJDK 11	Intel x86_64
SUSE Linux Enterprise Server 15 SP2 (x86_64) with OpenJDK 8	Intel x86_64
SUSE Linux Enterprise Server 15 SP4 (x86_64) with OpenJDK 8	Intel x86_64
SUSE Linux Enterprise Server 15 SP4 with OpenJDK 11 on ESXi 7.0.3	Dell PowerEdge R640
SUSE Linux Enterprise Server 15 SP4 with OpenJDK 17 on ESXi 7.0.3	Dell PowerEdge R640
SUSE Linux Enterprise Server 15 SP5 (x86_64) with OpenJDK 11	Intel x86_64
SUSE Linux Enterprise Server 15 SP5 (x86_64) with OpenJDK 17	Intel x86_64
SUSE Linux Enterprise Server 15 SP5 (x86_64) with OpenJDK 8	Intel x86_64
SUSE Linux Enterprise Server 15 SP6 with OpenJDK 11 on ESXi 7.0.3	Dell PowerEdge R640
SUSE Linux Enterprise Server 15 SP6 with OpenJDK 17 on ESXi 7.0.3	Dell PowerEdge R640
SUSE Linux Enterprise Server 15 SP6 with OpenJDK 21 on ESXi 7.0.3	Dell PowerEdge R640
SUSE Linux Enterprise Server 15 SP6 with OpenJDK 8 on ESXi 7.0.3	Dell PowerEdge R640
Windows Server 2019 with Oracle JRE 8 on ESXi 7.0.3	Dell PowerEdge R640
Windows Server 2022 with OpenJDK 11 on ESXi 7.0.3	Dell PowerEdge R640
Windows Server 2022 with OpenJDK 17 on ESXi 7.0.3	Dell PowerEdge R640

Table 4: Vendor-Affirmed Operational Environments - Software, Firmware, Hybrid

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.

2.3 Excluded Components

There are no components within the cryptographic boundary that are excluded from the module.

2.4 Modes of Operation

2.4.1 Modes List and Description:

Mode Name	Description	Type	Status Indicator
Approved	Approved Mode Of Operation	Approved	API CryptoModule.isFips140Mode() returns true

Table 5: Modes List and Description

When initialized per section 11.1, the module only supports the approved mode. To retrieve the current mode of operation, call:

- `FIPS140Context.getMode()`

A context value of `MODE_FIPS140` indicates that the module has been initialized correctly.

A context value of `MODE_NON_FIPS140` indicates that module has not been initialized correctly and will be offering cryptographic services that are not compliant with the requirements of this security policy.

2.4.2 Mode Change Instructions and Status:

The `ModuleLoader.load` API loads the module for use in the approved mode. This API returns the one and only instance of a `ModuleConfig` object.

The `ModuleConfig.newCryptoModule` API creates `CryptoModule` objects. This API supports a `FIPS140Context` parameter which specifies the mode of operation of the `CryptoModule`.

Refer to the API Javadoc for more information about these APIs.

2.4.3 Degraded Mode Description:

BSAFE Crypto Module does not support degraded operation. If any self-test fails, the cryptographic services of the module are disabled for both modes of operation.

2.5 Algorithms

2.5.1 Approved Algorithms:

The following table lists the BSAFE Crypto Module Approved algorithms, with the appropriate standards and CAVP validation certificate numbers:

Algorithm	CAVP Cert	Properties	Reference
AES-CBC	A6943	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CBC-CS1	A6943	Direction - decrypt, encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CBC-CS2	A6943	Direction - decrypt, encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CBC-CS3	A6943	Direction - decrypt, encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CCM	A6943	Key Length - 128, 192, 256	SP 800-38C
AES-CFB128	A6943	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CMAC	A6943	Direction - Generation, Verification Key Length - 128, 192, 256	SP 800-38B
AES-CTR	A6943	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-ECB	A6943	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-GCM	A6943	Direction - Decrypt, Encrypt IV Generation - External, Internal IV Generation Mode - 8.2.1 Key Length - 128, 192, 256	SP 800-38D
AES-KW	A6943	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38F
AES-KWP	A6943	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38F
AES-OFB	A6943	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-XTS Testing Revision 2.0	A6943	Direction - Decrypt, Encrypt Key Length - 128, 256	SP 800-38E
Counter DRBG	A6943	Prediction Resistance - No, Yes Mode - AES-128, AES-192, AES-256 Derivation Function Enabled - Yes	SP 800-90A Rev. 1

Algorithm	CAVP Cert	Properties	Reference
Deterministic ECDSA SigGen (FIPS186-5)	A6943	Curve - P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512 Component - No	FIPS 186-5
DSA KeyGen (FIPS186-4)	A6943	L - 2048, 3072 N - 224, 256	FIPS 186-4
DSA PQGGen (FIPS186-4)	A6943	L - 2048, 3072 N - 224, 256 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
DSA SigVer (FIPS186-4)	A6943	L - 1024, 2048, 3072 N - 160, 224, 256 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256	FIPS 186-4
ECDSA KeyGen (FIPS186-4)	A6943	Curve - B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571 Secret Generation Mode - Testing Candidates	FIPS 186-4
ECDSA KeyGen (FIPS186-5)	A6943	Curve - P-224, P-256, P-384, P-521 Secret Generation Mode - testing candidates	FIPS 186-5
ECDSA SigGen (FIPS186-4)	A6943	Component - No Curve - B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	FIPS 186-4
ECDSA SigGen (FIPS186-5)	A6943	Curve - P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512 Component - No	FIPS 186-5
ECDSA SigVer (FIPS186-4)	A6943	Component - No Curve - B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	FIPS 186-4
ECDSA SigVer (FIPS186-5)	A6943	Curve - P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	FIPS 186-5
EDDSA KeyGen	A6943	Curve - ED-25519, ED-448	FIPS 186-5
EDDSA SigGen	A6943	Curve - ED-25519, ED-448 PreHash - Yes Pure - Yes	FIPS 186-5
EDDSA SigVer	A6943	Curve - ED-25519, ED-448 PreHash - Yes Pure - Yes	FIPS 186-5
Hash DRBG	A6943	Prediction Resistance - No, Yes Mode - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256	SP 800-90A Rev. 1

Algorithm	CAVP Cert	Properties	Reference
HMAC DRBG	A6943	Prediction Resistance - No, Yes Mode - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256	SP 800-90A Rev. 1
HMAC-SHA-1	A6943	Key Length - Key Length: 8-51200 Increment 8	FIPS 198-1
HMAC-SHA2-224	A6943	Key Length - Key Length: 8-51200 Increment 8	FIPS 198-1
HMAC-SHA2-256	A6943	Key Length - Key Length: 8-51200 Increment 8	FIPS 198-1
HMAC-SHA2-384	A6943	Key Length - Key Length: 8-51200 Increment 8	FIPS 198-1
HMAC-SHA2-512	A6943	Key Length - Key Length: 8-51200 Increment 8	FIPS 198-1
HMAC-SHA2-512/224	A6943	Key Length - Key Length: 8-51200 Increment 8	FIPS 198-1
HMAC-SHA2-512/256	A6943	Key Length - Key Length: 8-51200 Increment 8	FIPS 198-1
HMAC-SHA3-224	A6943	Key Length - Key Length: 8-51200 Increment 8	FIPS 198-1
HMAC-SHA3-256	A6943	Key Length - Key Length: 8-51200 Increment 8	FIPS 198-1
HMAC-SHA3-384	A6943	Key Length - Key Length: 8-51200 Increment 8	FIPS 198-1
HMAC-SHA3-512	A6943	Key Length - Key Length: 8-51200 Increment 8	FIPS 198-1
KAS-ECC-SSC Sp800-56Ar3	A6943	Domain Parameter Generation Methods - B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 Scheme - ephemeralUnified - KAS Role - initiator, responder staticUnified - KAS Role - initiator, responder	SP 800-56A Rev. 3
KAS-FFC-SSC Sp800-56Ar3	A6943	Domain Parameter Generation Methods - FB, FC, ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 Scheme - dhEphem - KAS Role - initiator, responder dhOneFlow - KAS Role - initiator, responder dhStatic - KAS Role - initiator, responder	SP 800-56A Rev. 3
KAS-IFC-SSC	A6943	Modulo - 2048, 3072, 4096, 6144, 8192 Key Generation Methods - rsakpg1-basic Scheme - KAS1 - KAS Role - initiator, responder	SP 800-56A Rev. 3
KAS-KC SP800-56 (CVL)	A6943	KAS Role - Initiator, Responder Key Confirmation Methods - Key Confirmation Directions - Bilateral, Unilateral	SP 800-56A Rev. 3
KDA OneStep SP800-56Cr2	A6943	Derived Key Length - 2048 Shared Secret Length - Shared Secret Length: 224-8192 Increment 8	SP 800-56C Rev. 2

Algorithm	CAVP Cert	Properties	Reference
KDF SP800-108	A6943	KDF Mode - Feedback Supported Lengths - Supported Lengths: 8-4096 Increment 8	SP 800-108 Rev. 1
KTS-IFC	A6943	Modulo - 2048, 3072, 4096, 6144, 8192 Key Generation Methods - rsakpg2-basic Scheme - KTS-OAEP-basic - KAS Role - initiator, responder Key Transport Method - Key Length - 1024	SP 800-56B Rev. 2
LMS SigVer	A6943	LMS Modes - 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	SP 800-208
ML-DSA KeyGen	A6943	Parameter Sets - ML-DSA-44, ML-DSA-65, ML-DSA-87	FIPS 204
ML-DSA SigGen	A6943	Deterministic - No, Yes Signature Interfaces - external, internal Pre Hash - preHash, pure External Mu - No, Yes Parameter Sets - ML-DSA-44, ML-DSA-65, ML-DSA-87 Message Length - Message Length: 8-65536 Increment 8 Hash Algorithms - SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256 Context Length - Context Length: 0-2040 Increment 8	FIPS 204
ML-DSA SigVer	A6943	Signature Interfaces - external, internal Pre Hash - preHash, pure External Mu - No, Yes Parameter Sets - ML-DSA-44, ML-DSA-65, ML-DSA-87 Message Length - Message Length: 8-65536 Increment 8 Hash Algorithms - SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256 Context Length - Context Length: 0-2040 Increment 8	FIPS 204
PBKDF	A6943	Iteration Count - Iteration Count: 1-10000000 Increment 1 Password Length - Password Length: 8-128 Increment 1	SP 800-132
RSA KeyGen (FIPS186-5)	A6943	Key Generation Mode - probableWithProbableAux Modulo - 2048, 3072, 4096, 6144, 8192 Primality Tests - 2pow100 Private Key Format - crt	FIPS 186-5
RSA SigGen (FIPS186-5)	A6943	Modulo - 2048, 3072, 4096 Signature Type - pkcs1v1.5, pss	FIPS 186-5

Algorithm	CAVP Cert	Properties	Reference
RSA SigVer (FIPS186-4)	A6943	Signature Type - ANSI X9.31, PKCS 1.5, PKCSPSS Modulo - 1024, 2048, 3072, 4096	FIPS 186-4
RSA SigVer (FIPS186-5)	A6943	Modulo - 2048, 3072, 4096 Signature Type - pkcs1v1.5, pss	FIPS 186-5
Safe Primes Key Generation	A6943	Safe Prime Groups - ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	SP 800-56A Rev. 3
SHA-1	A6943	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-224	A6943	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-256	A6943	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-384	A6943	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-512	A6943	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-512/224	A6943	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-512/256	A6943	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA3-224	A6943	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHA3-256	A6943	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHA3-384	A6943	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHA3-512	A6943	Message Length - Message Length: 0-65536 Increment 8	FIPS 202
SHAKE-128	A6943	Output Length - Output Length: 16-65536 Increment 8	FIPS 202
SHAKE-256	A6943	Output Length - Output Length: 16-65536 Increment 8	FIPS 202
TLS v1.2 KDF RFC7627 (CVL)	A6943	Hash Algorithm - SHA2-256, SHA2-384, SHA2-512	SP 800-135 Rev. 1
TLS v1.3 KDF (CVL)	A6943	HMAC Algorithm - SHA2-256, SHA2-384 KDF Running Modes - DHE, PSK, PSK-DHE	SP 800-135 Rev. 1

Table 6: Approved Algorithms

2.5.2 Vendor-Affirmed Algorithms:

Name	Properties	Implementation	Reference
CKG	key type:Symmetric	BSAFE Crypto Module for Java	SP 800-133r2 section 4

Table 7: Vendor-Affirmed Algorithms

2.6 Security Function Implementations

Name	Type	Description	Properties	Algorithms
Authenticated Decryption	BC-Auth	Decryption using authenticated modes		AES-CCM: (A6943) AES-GCM: (A6943)
Authenticated Encryption	BC-Auth	Encryption using authenticated modes		AES-CCM: (A6943) AES-GCM: (A6943)
CKG	CKG	Symmetric key generation		CKG: ()

Name	Type	Description	Properties	Algorithms
Decapsulation	KTS-Encap	Asymmetric key decapsulation		KTS-IFC: (A6943) Function: Decrypt SHA-1: (A6943) Usage: MGF SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)
Decryption	BC-UnAuth	Decryption using unauthenticated modes		AES-CBC: (A6943) AES-CBC-CS1: (A6943) AES-CBC-CS2: (A6943) AES-CBC-CS3: (A6943) AES-CTR: (A6943) AES-ECB: (A6943) AES-XTS Testing Revision 2.0: (A6943) AES-CFB128: (A6943)
DetECDSA SigGen	DigSig-SigGen	Digital Signature Generation using deterministic ECDSA		Deterministic ECDSA SigGen (FIPS186-5): (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943) HMAC-SHA2-224: (A6943) HMAC-SHA2-256: (A6943) HMAC-SHA2-384: (A6943) HMAC-SHA2-512: (A6943) HMAC-SHA2-512/224: (A6943) HMAC-SHA2-512/256: (A6943)

Name	Type	Description	Properties	Algorithms
				HMAC-SHA3-224: (A6943) HMAC-SHA3-256: (A6943) HMAC-SHA3-384: (A6943) HMAC-SHA3-512: (A6943)
ECDSA SigGen	DigSig-SigGen	Digital Signature Generation using ECDSA		ECDSA SigGen (FIPS186-4): (A6943) ECDSA SigGen (FIPS186-5): (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)
ECDSA SigVer	DigSig-SigVer	Digital Signature Verification using ECDSA		ECDSA SigVer (FIPS186-5): (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)
EdDSA SigGen	DigSig-SigGen	Digital Signature Generation using EdDSA		EDDSA SigGen: (A6943) SHA2-512: (A6943) SHAKE-256: (A6943)
EdDSA SigVer	DigSig-SigVer	Digital Signature Verification using EdDSA		EDDSA SigVer: (A6943) SHA2-512: (A6943) SHAKE-256: (A6943)
Encapsulation	KTS-Encap	Asymmetric key encapsulation		KTS-IFC: (A6943) Function: Encrypt SHA-1: (A6943) Usage: MGF SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943)

Name	Type	Description	Properties	Algorithms
				SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)
Encryption	BC-UnAuth	Encryption using unauthenticated modes		AES-CBC: (A6943) AES-CBC-CS1: (A6943) AES-CBC-CS2: (A6943) AES-CBC-CS3: (A6943) AES-CFB128: (A6943) AES-CTR: (A6943) AES-ECB: (A6943) AES-XTS Testing Revision 2.0: (A6943) AES-OFB: (A6943)
KBKDF	KBKDF	Key-based key derivation		KDF SP800-108: (A6943) HMAC-SHA-1: (A6943) HMAC-SHA2-224: (A6943) HMAC-SHA2-256: (A6943) HMAC-SHA2-384: (A6943) HMAC-SHA2-512: (A6943) HMAC-SHA2-512/224: (A6943) HMAC-SHA2-512/256: (A6943) HMAC-SHA3-224: (A6943) HMAC-SHA3-256: (A6943) HMAC-SHA3-384: (A6943) HMAC-SHA3-512: (A6943) SHA-1: (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)

Name	Type	Description	Properties	Algorithms
KDF One Step	KAS-56CKDF	One step key derivation		KDA OneStep SP800-56Cr2: (A6943) SHA-1: (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)
KDF TLS 1.2	KAS-135KDF	Key derivation within TLS 1.2		TLS v1.2 KDF RFC7627: (A6943) HMAC-SHA2-256: (A6943) HMAC-SHA2-384: (A6943) HMAC-SHA2-512: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943)
KDF TLS 1.3	KBKDF	Key derivation within TLS 1.3		TLS v1.3 KDF: (A6943) HMAC-SHA2-256: (A6943) HMAC-SHA2-384: (A6943) SHA2-256: (A6943) SHA2-384: (A6943)
Key Confirmation	KAS-KC	Key Confirmation		HMAC-SHA-1: (A6943) HMAC-SHA2-224: (A6943) HMAC-SHA2-256: (A6943) HMAC-SHA2-384: (A6943) HMAC-SHA2-512: (A6943) HMAC-SHA2-512/224: (A6943) HMAC-SHA2-512/256: (A6943) HMAC-SHA3-224: (A6943) HMAC-SHA3-256: (A6943) HMAC-SHA3-512: (A6943) KAS-KC SP800-56: (A6943)
Key Generation	AsymKeyPair-KeyGen	Key Generation		DSA KeyGen (FIPS186-4): (A6943) ECDSA KeyGen (FIPS186-4): (A6943)

Name	Type	Description	Properties	Algorithms
				ECDSA KeyGen (FIPS186-5): (A6943) EDDSA KeyGen: (A6943) ML-DSA KeyGen: (A6943) RSA KeyGen (FIPS186-5): (A6943) Safe Primes Key Generation: (A6943)
Key Parameter Generation	AsymKeyPair-DomPar	Key Parameter Generation		DSA PQGGen (FIPS186-4): (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943)
Key Unwrap	KTS-Wrap	Perform Key Unwrapping		AES-KW: (A6943) AES-KWP: (A6943)
Key Wrap	KTS-Wrap	Perform Key Wrapping		AES-KW: (A6943) AES-KWP: (A6943)
LMS SigVer	DigSig-SigVer	Digital Signature Verification using LMS		LMS SigVer: (A6943) SHA2-256: (A6943) SHAKE-256: (A6943)
Legacy DSA SigVer	DigSig-SigVer	Digital Signature Verification using DSA		DSA SigVer (FIPS186-4): (A6943) SHA-1: (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943)
Legacy ECDSA SigVer	DigSig-SigVer	Digital Signature Verification using ECDSA		ECDSA SigVer (FIPS186-4): (A6943) SHA-1: (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)

Name	Type	Description	Properties	Algorithms
Legacy RSA SigVer	DigSig-SigVer	Digital Signature Verification using RSA		RSA SigVer (FIPS186-4): (A6943) SHA-1: (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)
MAC Generation	MAC	Perform MAC Generation		AES-CMAC: (A6943) HMAC-SHA-1: (A6943) HMAC-SHA2-224: (A6943) HMAC-SHA2-256: (A6943) HMAC-SHA2-384: (A6943) HMAC-SHA2-512: (A6943) HMAC-SHA2-512/224: (A6943) HMAC-SHA2-512/256: (A6943) HMAC-SHA3-224: (A6943) HMAC-SHA3-256: (A6943) HMAC-SHA3-384: (A6943) HMAC-SHA3-512: (A6943)
MAC Verification	MAC	Perform MAC Verification		AES-CMAC: (A6943) HMAC-SHA-1: (A6943) HMAC-SHA2-224: (A6943) HMAC-SHA2-256: (A6943) HMAC-SHA2-384: (A6943) HMAC-SHA2-512: (A6943) HMAC-SHA2-512/224: (A6943) HMAC-SHA2-512/256: (A6943) HMAC-SHA3-224: (A6943) HMAC-SHA3-256: (A6943) HMAC-SHA3-384: (A6943) HMAC-SHA3-512: (A6943)

Name	Type	Description	Properties	Algorithms
ML-DSA PureGen	DigSig-SigGen	Digital Signature Generation using Pure ML-DSA		ML-DSA SigGen: (A6943) SHAKE-128: (A6943) SHAKE-256: (A6943)
ML-DSA PreHashGen	DigSig-SigGen	Digital Signature Generation using Pre-Hash ML-DSA		ML-DSA SigGen: (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943) SHAKE-128: (A6943) SHAKE-256: (A6943)
ML-DSA PureVer	DigSig-SigVer	Digital Signature Verification using Pure ML-DSA		ML-DSA SigVer: (A6943) SHAKE-128: (A6943) SHAKE-256: (A6943)
ML-DSA PreHashVer	DigSig-SigVer	Digital Signature Verification using Pre-hash ML-DSA		ML-DSA SigVer: (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943) SHAKE-128: (A6943) SHAKE-256: (A6943)
Message Digest	SHA	Perform Message Digest		SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)

Name	Type	Description	Properties	Algorithms
PBKDF	PBKDF	Password-based key derivation		PBKDF: (A6943) HMAC-SHA-1: (A6943) HMAC-SHA2-224: (A6943) HMAC-SHA2-256: (A6943) HMAC-SHA2-384: (A6943) HMAC-SHA2-512: (A6943) HMAC-SHA2-512/224: (A6943) HMAC-SHA2-512/256: (A6943) HMAC-SHA3-224: (A6943) HMAC-SHA3-256: (A6943) HMAC-SHA3-384: (A6943) HMAC-SHA3-512: (A6943) SHA-1: (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)
RSA SigGen	DigSig-SigGen	Digital Signature Generation using RSA		RSA SigGen (FIPS186-5): (A6943) SHA-1: (A6943) Usage: MGF SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)
RSA SigVer	DigSig-SigVer	Digital Signature Verification using RSA		RSA SigVer (FIPS186-5): (A6943) SHA2-224: (A6943) SHA2-256: (A6943)

Name	Type	Description	Properties	Algorithms
				SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) SHA3-224: (A6943) SHA3-256: (A6943) SHA3-384: (A6943) SHA3-512: (A6943)
Random Number Generation	DRBG	Perform Random Number Generation		Counter DRBG: (A6943) Hash DRBG: (A6943) HMAC DRBG: (A6943) AES-CBC: (A6943) Function: Encrypt SHA-1: (A6943) SHA2-224: (A6943) SHA2-256: (A6943) SHA2-384: (A6943) SHA2-512: (A6943) SHA2-512/224: (A6943) SHA2-512/256: (A6943) HMAC-SHA-1: (A6943) HMAC-SHA2-224: (A6943) HMAC-SHA2-256: (A6943) HMAC-SHA2-384: (A6943) HMAC-SHA2-512: (A6943) HMAC-SHA2-512/224: (A6943) HMAC-SHA2-512/256: (A6943)
Shared Secret Computation	KAS-SSC	Computation of shared secrets		KAS-ECC-SSC Sp800-56Ar3: (A6943) KAS-FFC-SSC Sp800-56Ar3: (A6943) KAS-IFC-SSC: (A6943)
XOF	XOF	Extendable output function		SHAKE-128: (A6943) SHAKE-256: (A6943)

Table 8: Security Function Implementations

2.7 Algorithm Specific Information

2.7.1 AES-GCM

The module supports three methods of internal IV generation for AES-GCM:

- The four-byte salt derived from the TLS handshake process is input using the parameter `PARTIAL_IV` during cipher initialization. This is used as the first four bytes of IV. This 32-bit part of the IV is also referred to as the nonce value in FIPS140-3 IG C.H and is positioned in the name field of the IV as required by Scenario 3. The remaining eight bytes of IV, referred to as `nonce_explicit` in RFC5288, are generated deterministically by the module using the 64-bit counter used for the invocation field described above.
- IVs may be generated deterministically by not supplying any IV parameters during cipher initialization. The generated 96-bit (12-byte) IV consists of a 32-bit fixed field followed by a 64-bit invocation field. The fixed field bytes are derived from the module name, version information and memory address of a Java class within the module. The invocation field is a 64-bit counter that is initialized, on module startup, to a value consisting of the 42 bits of current time, as milliseconds since Epoch, followed by 22 bits of zero. This counter value is incremented by one each time a new IV is requested. By using the current time to prefix the counter start value, in the event of module restart, the counter will be ahead of any previous module states, ensuring that IV values cannot be reused. The module user must ensure the system time is valid to prevent repetition of IVs.
- At least 12 bytes of the IV may be generated from an Approved DRBG and provided to the cipher at initialization time using the `RAW_IV` parameter.

The Module does not persistently store AES-GCM keys or IVs. If the module's power is lost, new keys and IVs need to be generated. This condition is not enforced by the module but is met implicitly.

When used in the context of TLS, the IV for AES-GCM incorporates a 64-bit counter. When the 64-bit counter exhausts the maximum number of possible values for a given session key, the module will throw a `SecurityException`. Whichever party, the client or the server, that encounters this condition must trigger a handshake to establish a new encryption key.

When used in the context of TLS, the calling application must abort the TLS session if the keys for the client and server negotiated in the handshake process, `client_write_key` and `server_write_key`, are identical.

When using AES-GCM for symmetric encryption, the authentication tag length and authenticated data length may be specified as input parameters, but the IV must not be specified. The IV must be constructed from the output of an approved DRBG.

2.7.2 DSA Parameter Generation

When using an Approved DRBG to generate DH parameters using DSA.PQGGen, the requested DRBG must have a security strength at least as great as the security strength of the parameters being generated.

For more information on requesting the DRBG security strength, see the relevant API Javadoc.

2.7.3 DRBG seeding

When seeding the DRBG, the number of bits of entropy input must be equivalent to or greater than the security strength of the keys the caller wishes to generate. For example, when generating a 256-bit symmetric key, at least 256 bits of entropy input are required.

2.7.4 HMAC

The key length for an HMAC generation or verification must be between 112 and 51200 bits, inclusive.

For HMAC verification, a key length greater than or equal to 80 and less than 112 is allowed for legacy-use.

2.7.5 HMAC-Based Extract-and-Expand Key Derivation Function

A particular key-derivation key must only be used for a single key-expansion step. For more information see SP 800-56C Rev. 1.

The derived key must be used only as a secret key.

The derived key shall not be used as a key stream for a stream cipher.

When selecting an HMAC hash, the output block size must be equal to or greater than the desired security strength of the derived key.

The pseudo-random key input to the expansion and the keying material output from the expansion must have lengths that are equal to or greater than the desired security strength of the derived key.

2.7.6 One-Step Key Derivation Function

When selecting a hash algorithm, the output block size must be equal to or greater than the desired security strength of the derived key.

The derived key must be used only as a secret key.

The derived key shall not be used as a key stream for a stream cipher.

The secret data input into this KDF must have a length equal to or greater than the desired security strength of the derived key.

2.7.7 PBKDF

PBKDF can be used in the approved mode when used with an approved symmetric key and message digest algorithms.

Keys generated using PBKDF shall only be used in data storage applications.

The minimum length (L) of a password generated using a cryptographically secure random password generator to provide a search space of S entries depends on the size (N) of the character set:

$L = ceil(\log_2 S / \log_2 N)$

The following provides examples for a password used by PBKDF where $S = 2^{112} = 5.19 \times 10^{33}$:

Character Set	N	L
Case sensitive (a-z, A-Z)	52	20

Case sensitive alpha numeric	62	19
All ASCII printable characters except space	94	18

A password of the strength can be guessed at random with the probability of 1 in S.

The minimum length of the randomly generated portion of the salt is 16 bytes.

The iteration count is as large as possible, with a minimum of 10,000 iterations recommended.

The maximum key length is $(2^{32} - 1)*b$, where b is the digest size of the message digest function in bytes.

2.7.8 Shared secret computation

The module does not establish SSPs using an approved key agreement scheme (KAS). However, it does offer some or all of the underlying KAS cryptographic functionality to be used by an external operator/application as part of an approved KAS.

2.7.9 TLS Key Derivation Function

TLS1.2 KDF is allowed only when performed in the context of the TLS protocol.

For more information, see SP 800-135 Rev. 1.

The TLS protocols have not been tested by the CAVP and CMVP.

2.7.10 XTS-AES

For XTS-AES keys that are entered into the module, the operator must ensure that the two halves of the key (payload and tweak) have been generated and/or established independently of each other, according to the rules for component symmetric keys from NIST SP 800-133rev2, section 6.3.

For XTS-AES keys generated by the module, this independence requirement is implicitly met.

XTS-AES is approved only for hardware storage applications.

2.7.11 Legacy functions

Security function implementations designated as "Legacy" can only be used on data that was generated prior to the Legacy Date specified in FIPS 140-3 IG C.M.

2.8 RBG and Entropy

The BSAFE Crypto Module does not provide an entropy source. The module is passively receiving the entropy while exercising no control over the amount or the quality of the obtained entropy. This conforms to FIPS 140-3 IG 9.3.A scenario 2b. The following entropy caveat applies: *"No assurance of the minimum strength of generated SSPs."*

It is the operator's responsibility to supply the entropy to seed a DRBG to provide the required security strength, and to ensure the security strength of the DRBG is equal to or greater than the security strength of any SSPs generated using that DRBG.

Entropy can be supplied to the module using the following APIs:

- `com.rsa.crypto.SecureRandom.setSeed()`
- `com.rsa.crypto.ModuleConfig.setEntropySource()`

The JCM provides the HMAC DRBG, CTR DRBG and Hash DRBG implementations for SSP generation.

When generating SSPs, the DRBG used in key generation must be seeded with a number of bits of entropy that is equal to or greater than the security strength of the SSP being generated. The entropy supplied to the DRBG is referred to as the DRBG security strength which represents the minimum amount of entropy that should be provided to the DRBG prior to generating the SSP.

In the `DefaultEntropySource` class the module uses the following default seeders in this priority order (if not specified using the public APIs):

- Sun MSCAPI on Windows platforms
- SUN on Oracle JRE platforms
- IBMJCE on IBM platforms

2.9 Key Generation

Symmetric keys	• The module performs Cryptographic Key Generation (CKG) in compliance with section 4 of SP 800-133r2. Symmetric cryptographic keys are generated by the direct unmodified output of the module's approved DRBG.
FFC (DH) parameter generation	• As specified in FIPS 186-4.
DH key pair generation	• FIPS 186-4 Appendix B.1.1 Key Pair Generation by Using Extra Random Bits
ECDSA key pair generation	• FIPS 186-5 Appendix A.2.2 Key Pair Generation by Testing Candidates
RSA key pair generation	• FIPS 186-5 Appendix A.1.6 Step 6 • FIPS 186-5 Appendix A.1.1 method B case 3
Prime number processing	• Miller-Rabin prime number testing • FIPS 186-5 Appendix B.3.2 Enhanced Miller-Rabin test
RSA public key validation	• As specified in SP 800-89, Section 5.3.3.
RSASVE	• As specified in SP 800-56B Revision 2.
RSA PKCS #1 PSS signature	• Message encoding.
ML-DSA key pair generation	• As specified in FIPS 204

Table 9: DRBG Output Uses

The following table lists each of the keys that can be generated by the JCM, with the key sizes available, security strengths for each key size and the security strength required to initialize the DRBG.

Key Type	Key Size	Security Strength	Required DRBG Security Strength
AES Key	128, 192, 256	128, 192, 256	128, 192, 256
RSA Key Pair	2048, 3072, 4096, 6144, 8192	112, 128, 192	112, 128, 192
DH Key Pair	2048, 3072, 4096, 6144, 8192	112, 128, 192	112, 128, 192
EC Key Pair	P-224, P-256, P-384, P-521	112, 128, 192, 256	112, 128, 192, 256
EdDSA Key Pair	Edwards25519, Edwards448	128, 224	128, 224
ML-DSA Key Pair	Private key (2560, 4032 and 4896 byte key sizes) Public key (1312, 1952 and 2592 byte key sizes)	Category 2, 3 and 5[1]	128, 192, 256

Table 10: Generated Key Sizes and Strength

Security strength categories as defined in by NISTs PQC Security evaluation criteria.

2.10 Key Establishment

2.10.1 Key Agreement

The module does not establish SSPs using an approved key agreement scheme (KAS). However, it does offer some or all of the underlying KAS cryptographic functionality to be used by an external operator/application as part of an approved KAS.

- The module establishes a DH shared secret compliant to SP 800-56Arev3.
- The module establishes an EC DH shared secret compliant to SP 800-56Arev3.
- The module establishes an IFC shared secret compliant to SP 800-56Brev2.

2.10.2 Key Transport

The module does not establish SSPs using an approved key transport scheme (KTS). However, it does offer approved authenticated algorithms that can be used by an external operator/application as part of an approved KTS.

- The module performs key wrapping using authentication encryption modes of AES (KW and KWP) compliant to SP 800-38F. This SSP establishment methodology provides between 112 and 256 bits of encryption strength.
- The module provides key transport using a KTS-OAEP-basic scheme compliant to SP 800-56B rev 2. This SSP establishment methodology provides between 112 and 256 bits of encryption strength.

2.11 Industry Protocols

No part of the TLS protocol, other than the approved cryptographic algorithms and the KDF, have been tested by the CAVP and CMVP.

3 Cryptographic Module Interfaces

BSAFE Crypto Module is a software module that provides APIs only as logical interfaces. Physical ports and interfaces are not provided by the module.

The module conforms to the FIPS 140-3 Security Level 1 requirements for Cryptographic Module Interfaces

3.1 Ports and Interfaces

The following table lists the ports and interfaces, and the data that passes over each:

Physical Port	Logical Interface(s)	Data That Passes
N/A	Control Input	Configuration parameters for the API interface Module Config which sets the mode of operation.
N/A	Data Input	Plaintext, Ciphertext, Message Digest, Signature, MAC, Secret Keys, Key Text, Wrapped Key Text
N/A	Data Output	Status, ciphertext, plaintext, signature, key text, wrapped key text, MAC, Random bytes
N/A	Status Output	Mode of operation indicator from the API CryptoModule.isFIPS140Mode(). The state of the module from the API CryptoModule.getState().

Table 11: Ports and Interfaces

The module does not implement a control output interface.

4 Roles, Services, and Authentication

BSAFE Crypto Module meets all FIPS 140-3 Security Level 1 requirements for Roles, Services and Authentication, implementing a Crypto Officer Role. As allowed by FIPS 140-3, the module does not support identification or authentication for this role. There is no maintenance role. The module does not allow concurrent operators.

4.1 Authentication Methods

The module does not implement authentication. The Crypto Officer Role is implicitly assumed once the module is loaded and cleared on module unload.

4.2 Roles

Name	Type	Operator Type	Authentication Methods
Crypto Officer	Role	Crypto Officer	None

Table 12: Roles

The module supports a single role, denoted as Crypto Officer. This role is responsible for installing and loading the module. Once the module has been installed and is operational, the operator assumes the Crypto Officer Role by constructing a `com.rsa.crypto.FIPS140Context` object by invoking the `ModuleConfig.getFIPS140Context(int mode, int role)` method with a role argument of `com.rsa.crypto.FIPS140Context.ROLE_CRYPTO_OFFICER`.

4.3 Approved Services

The following table lists the services provided by the JCM

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Authenticated decryption	Perform authenticated decryption	API return value	Ciphertext, MAC, key	Plaintext, status	Authenticated Decryption	Crypto Officer - AES Key: W,E - AES GCM IV: W,E
Authenticated encryption	Perform authenticated encryption	API return value	Plaintext, key	Ciphertext, MAC, Status	Authenticated Encryption	Crypto Officer - AES Key: W,E - AES GCM IV: W,E
Decapsulation	Asymmetric key decapsulation	API return value	Ciphertext, Key	Plaintext, Status	Decapsulation	Crypto Officer - RSA Private key: W,E
Decryption	Perform unauthenticated decryption	API return value	Ciphertext, key	Plaintext, Status	Decryption	Crypto Officer - AES Key: W,E
Digital Signature Generation	Digital Signature Generation	API return value	Message Digest	Signature, Status	DetECDSA SigGen ECDSA SigGen EdDSA SigGen ML-DSA PureGen ML-DSA PreHashGen RSA SigGen	Crypto Officer - EC Private Key: W,E - EdDSA Private key: W,E - ML-DSA Private Key: W,E - RSA Private key: W,E
Digital Signature Verification	Digital Signature Verification	API return value	Message Digest, Signature	Verify Status, Status	ECDSA SigVer EdDSA SigVer LMS SigVer ML-DSA PureVer ML-DSA PreHashVer RSA SigVer Legacy DSA SigVer Legacy ECDSA SigVer Legacy RSA SigVer	Crypto Officer - DSA Public key: W,E - EC Public key: W,E - EdDSA Public key: W,E - LMS Public key: W,E - ML-DSA Public Key: W,E - RSA Public key: W,E
Encapsulation	Asymmetric key encapsulation	API return value	Plaintext, Key	Ciphertext, Status	Encapsulation	Crypto Officer - RSA Private key: W,E
Encryption	Perform unauthenticated encryption	API return value	Plaintext, key	Ciphertext, status	Encryption	Crypto Officer - AES Key: W,E
Key Assurance	Perform explicit validation of public and private keys	API return value	Keys	Validation Status, Status	None	Crypto Officer - RSA Private key: W - RSA Public key: W
Key Confirmation	Key Confirmation	API return value	MAC	verify Status	Key Confirmation	Crypto Officer - HMAC key: W,E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Key Derivation	Key Derivation	API return value	Secret	Key Text, Status	KBKDF KDF One Step KDF TLS 1.2 KDF TLS 1.3 PBKDF	Crypto Officer - HKDF key derivation key: W,E - HKDF derived key: G,R - PBKDF2 derived key: G,R - PBKDF2 Password: W,E - One-Step KDF Secret: W,E - One-Step KDF derived key: G,R - TLS 1.3 KDF derived key: G,R - TLS 1.2 KDF derived key: G,R - TLS Master secret: W,E
Key Generation	Perform Key Generation	API return value	Domain parameters	Key pair, Status	Key Generation CKG	Crypto Officer - DH domain parameters: W,E - DH Public key: G - DH Private key: G - EC Private Key: G - EC Public key: G - EdDSA Private key: G - EdDSA Public key: G - RSA Public key: G - RSA Private key: G - ML-DSA Public Key: G - ML-DSA Private Key: G - AES Key: G - CMAC key: G - HMAC key: G
Key Parameter Generation	Perform Key Parameter Generation	API return value	Random number	Domain parameters, status	Key Parameter Generation	Crypto Officer - DH domain parameters: G,R

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Key Unwrap	Perform Key Unwrap	API return value	Ciphertext, key	Plaintext, Status	Key Unwrap	Crypto Officer - AES key wrap key: W,E
Key Wrap	Perform Key Wrap	API return value	Plaintext, key	Ciphertext, Status	Key Wrap	Crypto Officer - AES key wrap key: W,E
MAC Generation	Perform MAC Generation	API return value	Plaintext, key	MAC, Status	MAC Generation	Crypto Officer - HMAC key: W,E - CMAC key: W,E
MAC Verification	Perform MAC Verification	API return value	Plaintext, ciphertext, key	Status	MAC Verification	Crypto Officer - HMAC key: W,E - CMAC key: W,E
Message Digest	Perform Message Digest Operation	API return value	Message	Message Digest, Status	Message Digest XOF	Crypto Officer
Random Number Generation	Perform Random Number Generation	API return value	Entropy input	Random bytes, Status	Random Number Generation	Crypto Officer - Hash DRBG Seed: G,E - Hash DRBG V: G,E - Hash DRBG C: G,E - HMAC DRBG Seed: G,E - HMAC DRBG V: G,E - HMAC DRBG Key: G,E - CTR DRBG Seed: G,E - CTR DRBG V: G,E - CTR DRBG Key: G,E - Entropy input: W
Self-test	Perform Self-test	API return value	Command	Status	None	Unauthenticated
Shared secret computation	Computer a shared secret	API return value	Keys, domain parameters	Shared secret, status	Shared Secret Computation	Crypto Officer - DH Private key: W,E - DH Public key: W,E - DH Shared Secret: G,R - DH domain parameters: W,E - EC Private Key: W,E - EC Public key: W,E - EC Shared Secret: G,R
Show status	Show module status	-	Command	Status	None	Crypto Officer

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Show version	Show module name and version	-	Command	Status	None	Crypto Officer
Zeroisation	Perform SSP zeroisation	-	-	-	None	Unauthenticated

Table 13: Approved Services

In the BSAFE Crypto Module all cryptographic objects and services are created by a `com.rsa.crypto.CryptoModule`. This can be created within the context of a `com.rsa.crypto.FIPS140Context`.

When a `com.rsa.crypto.CryptoModule` is created with a `com.rsa.crypto.FIPS140Context` it inherits its mode, and it is fixed in that mode for its lifetime. The `com.rsa.crypto.CryptoModule.isFIPS140Mode` method provides access to the mode.

All cryptographic services implement interfaces that extend the `com.rsa.crypto.CryptoObject` interface. This interface defines the `getCryptoModule()` method that provides a reference to the `com.rsa.crypto.CryptoModule` which created the service. The mode indicator for each service is provided by the `isFIPS140Mode()` method of the `com.rsa.crypto.CryptoModule` referenced by the service.

All cryptographic objects and services either implement the `com.rsa.crypto.FIPS140Indicator` interface to provide an `isFIPS140()` method or the `com.rsa.crypto.FIPS140IndicatorOperation` interface to provide an `isFIPS140(Operation operation)` method. These methods can be used to query if the cryptographic algorithm is approved for use in an approved mode of operation.

For more information on each function, see the relevant API Javadoc.

The module does not persistently store SSPs and does not support an invocable zeroisation service. SSPs are zeroised by reloading the module.

4.4 Non-approved services

The module does not support a non-approved mode and does not offer any non-approved services.

4.5 External Software Loaded

N/A. The module does not support the loading of software.

5 Software Security

This section covers integrity measures to demonstrate protection of the software component of BSAFE Crypto Module, which is the whole of the module. The integrity tests used are described in detail in Self-tests.

5.1 Integrity Techniques

The BSAFE Crypto Module start-up integrity check is implemented by first calculating a MAC over each of the files listed in `module.files`, using HMAC-SHA-1 with a fixed key.

Another MAC is then calculated over all file MACs in the order that they are listed, using the same algorithm and key as for the file MACs. This two-step process is intended to allow the jar file to be processed sequentially without having to load the entire jar file into memory even after the order of the jar file entries has been changed. The expected integrity check MAC is stored in the jar file manifest.

During the Integrity Test when the BSAFE Crypto Module is loaded, a MAC is again calculated and compared with the pre-computed MAC value contained in the jar file manifest. If these values are equal then the software integrity check has passed and power-up of the module can continue. Otherwise, the test has failed and the module is disabled

5.2 Initiate on Demand

The module provides the `ModuleConfig.runSelfTests()` API to allow the operator to perform on-demand tests. This API runs the integrity tests after all CAST's have completed successfully.

6 Operational Environment

BSAFE Crypto Module is FIPS 140-3 validated to operate in a modifiable environment.

The module is provided for operating systems running on a general purpose computer platform based on an Intel CPU.

The following steps describe how to instantiate the module in the approved mode of operation:

- If required, set the `com.rsa.cryptoj.eventhandler` property for logging FIPS 140-3 self-test events.
- Call the BSAFE Crypto Module `ModuleLoader.load` method which returns the `JavaModuleConfig` singleton.
- Call `JavaModuleConfig.newCryptoModule` to obtain an instance of `JavaCryptoModule`.
- The factory methods on `JavaCryptoModule` can then be used to perform cryptographic functions and access cryptographic services.

6.1 Operational Environment Type and Requirements

Type of Operational Environment: Modifiable

6.2 Configuration Settings and Restrictions

The module runs on a general purpose computer running one of the operating environments listed in Tested Operational Environments and Vendor Affirmed Operational Environments.

Each supported operating environment manages its own processes and memory in a logically separated manner. The process management setting is not configurable on the supported operating environments.

6.3 Additional Information

Each instance of the module that is loaded within an operating system maintains its own instance of internal SSPs. Any additional SSPs loaded into a given instance of the module are not available to other instances.

The supported operating environments provide process isolation, with resource and memory protection. Each instance of the module is isolated from others such that SSPs can only be accessed or modified in the module to which they belong.

BSAFE Crypto Module does not spawn additional processes.

7 Physical Security

BSAFE Crypto Module is classified as a multi-chip standalone cryptographic module. The module is comprised of software only, and does not claim any physical security.

8 Non-Invasive Security

BSAFE Crypto Module does not implement any non-invasive mitigation techniques.

9 Sensitive Security Parameters Management

9.1 Storage Areas

Storage Area Name	Description	Persistence Type
RAM	Volatile memory	Dynamic

Table 14: Storage Areas

9.2 SSP Input-Output Methods

Name	From	To	Format Type	Distribution Type	Entry Type	SFI or Algorithm
From App	Calling application	RAM	Plaintext	Automated	Electronic	
To App	RAM	Calling application	Plaintext	Automated	Electronic	

Table 15: SSP Input-Output Methods

9.3 SSP Zeroization Methods

Zeroization Method	Description	Rationale	Operator Initiation
Auto	Automatic zeroisation of SSPs that are no longer needed.	Memory is overwritten with 0's	N/A
Reboot	Power-cycle the host device	All RAM is zeroised by a power-cycle.	Unplugging the host device
Zcmd	The module stores all its keys and CSPs in volatile memory. Users can ensure CSPs are properly zeroised by making use of the clearSensitiveData() method.	All of the module's keys and CSPs are zeroisable. The clearSensitiveData() method overwrites the key or CSP value with zeros.	API call to the clearSensitiveData() method

Table 16: SSP Zeroization Methods

The module stores SSPs in the volatile memory of the host device. To completely zeroize all SSPs, the operator must reboot the host device. The operator must be present to observe that the zeroization method has been completed successfully.

Temporary SSPs are zeroized immediately after they are no longer in use. Other SSPs are zeroized when the associated key or cryptographic object is deleted.

Protection of the SSPs in volatile memory is provided by the operating environment which isolates the memory of separate processes.

9.4 SSPs

The following tables list the SSPs present in the module and details of how they are used and accessed:

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
AES GCM IV	-	96 - -	IV - CSP	TLS v1.2 KDF RFC7627 (A6943) TLS v1.3 KDF (A6943)		Authenticated Decryption Authenticated Encryption
AES Key	-	128, 192, 256 - 128, 192, 256	Symmetric - CSP	CKG KBKDF KDF One Step KDF TLS 1.2 KDF TLS 1.3 PBKDF		Decryption Encryption
AES key wrap key	-	128, 192, 256 - 128, 192, 256	Symmetric - CSP			Key Wrap Key Unwrap
CMAC key	-	128, 192, 256 - 128, 192, 256	Symmetric - CSP	CKG		MAC Generation MAC Verification
CTR DRBG Key	-	128, 192, 256 - -	DRBG State - CSP	Random Number Generation		Random Number Generation
CTR DRBG Seed	-	256, 320, 384 - -	DRBG State - CSP	Random Number Generation		Random Number Generation
CTR DRBG V	-	128 - -	DRBG State - CSP	Random Number Generation		Random Number Generation
DH Private key	-	2048, 3072 - 112, 128	Asymmetric - CSP	Key Generation		Shared Secret Computation
DH Public key	-	2048, 3072 - 112, 128	Asymmetric - PSP	Key Generation		Shared Secret Computation
DH Shared Secret	-	112, 128 - 112, 128	Shared Secret - CSP		Shared Secret Computation	KDF TLS 1.2 KDF TLS 1.3
DH domain parameters	-	- - -	Domain parameters - Neither	Key Parameter Generation Shared Secret Computation		Key Generation
DSA Public key	-	2048, 3072 - 112, 128	Asymmetric - PSP			Legacy DSA SigVer
EC Private Key	-	P-224, P-256, P-384, P-521, B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571 - 112, 128, 192, 256	Asymmetric - CSP	Key Generation		DetECDSA SigGen ECDSA SigGen Shared Secret Computation

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
EC Public key	-	P-224, P-256, P-384, P-521, B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571 - 112, 128, 192, 256	Asymmetric - PSP	Key Generation		ECDSA SigVer Legacy ECDSA SigVer Shared Secret Computation
EC Shared Secret	-	224, 256, 384, 521 - 224, 256, 384, 512	Shared Secret - CSP		Shared Secret Computation	KDF TLS 1.2 KDF TLS 1.3
EdDSA Private key	-	128, 224 - 128, 224	Asymmetric - CSP	Key Generation		EdDSA SigGen
EdDSA Public key	-	128, 224 - 128, 224	Asymmetric - PSP	Key Generation		EdDSA SigVer
Entropy input	-	112, 256 - 112, 256	Entropy - CSP			Random Number Generation
HKDF derived key	-	2048 - 160, 224, 256, 384, 512	Symmetric - CSP	KBKDF		
HKDF key derivation key	-	112 to 4096 - 112 to 512	Symmetric - CSP			KBKDF
HMAC DRBG Key	-	160, 224, 256, 384, 512 - -	DRBG State - CSP	Random Number Generation		Random Number Generation
HMAC DRBG Seed	-	440, 888 - -	DRBG State - CSP	Random Number Generation		Random Number Generation
HMAC DRBG V	-	160, 224, 256, 384, 512 - -	DRBG State - CSP	Random Number Generation		Random Number Generation
HMAC key	-	112, 128, 192, 256 - 112, 128, 192, 256	Symmetric - CSP	CKG KDF TLS 1.2		MAC Generation MAC Verification
Hash DRBG C	-	440, 888 - -	DRBG State - CSP	Random Number Generation		Random Number Generation
Hash DRBG Seed	-	440, 888 - -	DRBG State - CSP	Random Number Generation		Random Number Generation
Hash DRBG V	-	440, 888 - -	DRBG State - CSP	Random Number Generation		Random Number Generation
LMS Public key	-	384, 448 - 192, 256	Asymmetric - PSP			LMS SigVer
ML-DSA Private Key	-	2560, 4032, 4896 bytes - 128, 192, 256	Asymmetric - CSP	Key Generation		ML-DSA PureGen ML-DSA PreHashGen
ML-DSA Public Key	-	1312, 1952, 2592 bytes - 128, 192, 256	Asymmetric - PSP	Key Generation		ML-DSA PureVer ML-DSA PreHashVer
One-Step KDF Secret	-	224 to 8192 - -	Secret - CSP			KDF One Step
One-Step KDF derived key	-	160 to 2048 - 112 to 512	Symmetric - CSP	KDF One Step		

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
PBKDF2 Password	-	8 to 128 - -	Password - CSP			PBKDF
PBKDF2 derived key	-	112 to 4096 - -	Symmetric - CSP	PBKDF		
RSA Private key	-	2048, 3072, 4096 - 112, 128, 152	Asymmetric - CSP	Key Generation		Decapsulation RSA SigGen
RSA Public key	-	2048, 3072, 4096 - 112, 128, 152	Asymmetric - PSP	Key Generation		RSA SigVer Legacy RSA SigVer Encapsulation
TLS 1.2 KDF derived key	-	256, 384, 512 - 256, 384, 512	Keying material - CSP	TLS v1.2 KDF RFC7627 (A6943)		
TLS 1.3 KDF derived key	-	256, 384 - 256, 384	Keying material - CSP	TLS v1.3 KDF (A6943)		
TLS Master secret	-	112, 128, 192, 256 - 112, 128, 192, 256	Shared secret - CSP	KDF TLS 1.2 KDF TLS 1.3		KDF TLS 1.2 KDF TLS 1.3

Table 17: SSP Table 1

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
AES GCM IV	To App From App	RAM:Plaintext	-	Zcmd Reboot	
AES Key	To App From App	RAM:Plaintext	-	Zcmd Reboot	
AES key wrap key	From App	RAM:Plaintext	-	Zcmd Reboot	
CMAC key	From App	RAM:Plaintext	-	Zcmd Reboot	
CTR DRBG Key		RAM:Plaintext	-	Zcmd Reboot	
CTR DRBG Seed		RAM:Plaintext	-	Auto	Entropy input:Derived From
CTR DRBG V		RAM:Plaintext	-	Zcmd Reboot	
DH Private key	To App From App	RAM:Plaintext	-	Zcmd Reboot	DH Public key:Paired With
DH Public key	To App From App	RAM:Plaintext	-	Zcmd Reboot	DH Private key:Paired With
DH Shared Secret	To App	RAM:Plaintext	-	Zcmd Reboot	DH Private key:Derived From DH Public key:Derived From
DH domain parameters	To App	RAM:Plaintext	-	Zcmd Reboot	

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
DSA Public key	From App	RAM:Plaintext	-	Zcmd Reboot	
EC Private Key	To App From App	RAM:Plaintext	-	Zcmd Reboot	EC Public key:Paired With
EC Public key	To App From App	RAM:Plaintext	-	Zcmd Reboot	EC Private Key:Paired With
EC Shared Secret	To App	RAM:Plaintext	-	Zcmd Reboot	EC Private Key:Derived From EC Public key:Derived From
EdDSA Private key	To App From App	RAM:Plaintext	-	Zcmd Reboot	EdDSA Public key:Paired With
EdDSA Public key	To App From App	RAM:Plaintext	-	Zcmd Reboot	EdDSA Private key:Paired With
Entropy input	From App	RAM:Plaintext	-	Zcmd Reboot	
HKDF derived key	To App	RAM:Plaintext	-	Zcmd Reboot	HKDF key derivation key:Derived From
HKDF key derivation key	From App	RAM:Plaintext	-	Zcmd Reboot	
HMAC DRBG Key		RAM:Plaintext	-	Zcmd Reboot	
HMAC DRBG Seed		RAM:Plaintext	-	Auto	Entropy input:Derived From
HMAC DRBG V		RAM:Plaintext	-	Zcmd Reboot	
HMAC key	To App From App	RAM:Plaintext	-	Zcmd Reboot	
Hash DRBG C		RAM:Plaintext	-	Zcmd Reboot	Entropy input:Derived From
Hash DRBG Seed		RAM:Plaintext	-	Auto	
Hash DRBG V		RAM:Plaintext	-	Zcmd Reboot	
LMS Public key	From App	RAM:Plaintext	-	Zcmd Reboot	
ML-DSA Private Key	To App From App	RAM:Plaintext	-	Zcmd Reboot	ML-DSA Public Key:Paired With
ML-DSA Public Key	To App From App	RAM:Plaintext	-	Zcmd Reboot	ML-DSA Private Key:Paired With
One-Step KDF Secret	From App	RAM:Plaintext	-	Zcmd Reboot	
One-Step KDF derived key	To App	RAM:Plaintext	-	Zcmd Reboot	One-Step KDF Secret:Derived From
PBKDF2 Password	From App	RAM:Plaintext	-	Zcmd Reboot	

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
PBKDF2 derived key	To App	RAM:Plaintext	-	Zcmd Reboot	PBKDF2 Password:Derived From
RSA Private key	To App From App	RAM:Plaintext	-	Zcmd Reboot	
RSA Public key	To App From App	RAM:Plaintext	-	Zcmd Reboot	
TLS 1.2 KDF derived key	To App	RAM:Plaintext	-	Zcmd Reboot	TLS Master secret:Derived From
TLS 1.3 KDF derived key	To App	RAM:Plaintext	-	Zcmd Reboot	TLS Master secret:Derived From
TLS Master secret	From App To App	RAM:Encrypted	-	Zcmd Reboot	DH Shared Secret:Derived From EC Shared Secret:Derived From

Table 18: SSP Table 2

Keys are imported and exported in plain text using the Module's data input/output APIs.

The SSPs can be generated by the SP 800-90A DRBG.

9.5 Transitions

The module implements the following security methods that are forecasted to transition to non-approved over the lifetime of the validation:

Algorithm	Uses
SHA-1	As the MGF (Mask Generation Function) in RSA algorithms
SHA2-224	For hashing In RSA signature generation In ECDSA signature generation In ML-DSA signature generation In RSA-based key encapsulation (KTS-IFC) In DSA domain parameter generation (PQGGen) As the auxiliary function in Hash_DRBG
SHA3-224	For hashing In RSA signature generation In ECDSA signature generation In ML-DSA signature generation In RSA-based key encapsulation (KTS-IFC)
HMAC-SHA-1	For MAC Generation As the PRF in PBKDF As the auxiliary function in HMAC_DRBG
HMAC-SHA2-224	For MAC Generation As the PRF in PBKDF As the PRF in KBKDF As the auxiliary function in HMAC_DRBG
HMAC-SHA3-224	For MAC Generation As the PRF in PBKDF As the PRF in KBKDF

No algorithm or security strength transitions are forecasted to occur over the lifetime of the validation.

10 Self-Tests

BSAFE Crypto Module performs a number of pre-operational and conditional self-tests to ensure proper operation.

If a self-test fails, all cryptographic services for the library are disabled.

Cryptographic services for the module can be re-enabled only by reloading the module.

Any self-test failure causes the module to transition into the `FIPS140State.FAILED` state, throwing a `SecurityException`.

10.1 Pre-Operational Self-Tests

The following table specifies the pre-operational self-tests implemented in the module:

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details
Integrity test	160-bit hash, 240-bit key	KAT	SW/FW Integrity	API return code	-

Table 19: Pre-Operational Self-Tests

Pre-operational self-tests are executed automatically when the module is loaded into memory. They can be rerun manually after the module has loaded by calling the `ModuleConfig.runSelfTests()` API.

The pre-operational self-tests include the Software Integrity Test. The Software Integrity Test is comprised of an HMAC-SHA-1 verification of the files listed in `fips140/module.files`.

The cryptographic services of the module are disabled when the self-tests are running.

- All cryptographic operations throw a `CryptoException`
- The `CryptoModule.getState()` status output interface returns a state of `com.rsa.crypto.FIPS140State.UNDER_SELF_TEST`.

If any pre-operational self-test fails, all cryptographic services of the module are disabled.

- All cryptographic operations throw a `CryptoException`
- The `CryptoModule.getState()` status output interface returns a state of `com.rsa.crypto.FIPS140State.FAILED`.

If the pre-operational self-tests pass, the cryptographic services of the module are enabled and the module can be used. The `CryptoModule.getState()` status output interface returns a state of `com.rsa.crypto.FIPS140State.OPERATIONAL`.

10.2 Conditional Self-Tests

The following table specifies the conditional self-tests implemented in the module:

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
AES CCM	128-bit key	KAT	CAST	API return code	Encrypt, decrypt	Module load
AES CMAC	128-bit key	KAT	CAST	API return code	Generate	Module load
AES ECB	128-bit key	KAT	CAST	API return code	Encrypt, decrypt	Module load
AES GCM	128-bit key	KAT	CAST	API return code	Encrypt, decrypt	Module load
Counter DRBG	128-bit key	KAT	CAST	API return code	Instantiate, Generate, Reseed	Module load
DH DSA key pairs	-	PCT	PCT	API return code	Sign and verify	Key Pair Generation
DSA Verify	L=2048, N=224, SHA2-256	KAT	CAST	API return code	Verify	Module load
Diffie-Hellman	2048-bit FFC shared secret calculation	KAT	CAST	API return code	Calculate	Module load
ECDH	P-256, ECC shared secret calculation	KAT	CAST	API return code	Calculate	Module load
ECDSA Sign	P-256, B-233, fixed k-value	KAT	CAST	API return code	Sign	Module load
ECDSA Verify	P-256, B-233	KAT	CAST	API return code	Verify	Module load
ECDSA key pairs	-	PCT	PCT	API return code	Sign and verify	Key Pair Generation
EdDSA Sign	Edwards25519, Edwards448	KAT	CAST	API return code	Sign	Module load
EdDSA Verify	Edwards25519, Edwards448	KAT	CAST	API return code	Verify	Module load
EdDSA key pairs	-	PCT	PCT	API return code	Sign and verify	Key Pair Generation
HMAC DRBG	SHA-1	KAT	CAST	API return code	Instantiate, Generate, Reseed	Module load
HSS Verify	LMS_SHA256_M32_H5	KAT	CAST	API return code	Verify	Module load
Hash DRBG	SHA-1	KAT	CAST	API return code	Instantiate, Generate, Reseed	Module load

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
KDA OneStep	SHA2-224, 2048 bits	KAT	CAST	API return code	Generate	Module load
ML-DSA Key pairs	-	PCT	PCT	API return code	Sign and verify	Key Pair Generation
ML-DSA KeyGen	ML-DSA-44	KAT	CAST	API return code	Generate	Module load
ML-DSA Sign	ML-DSA-65	KAT	CAST	API return code	Sign	Module load
ML-DSA Verify	ML-DSA-87	KAT	CAST	API return code	Verify	Module load
PBKDF	SHA-1, 10,000 iterations	KAT	CAST	API return code	Derive	Module load
RSA Cipher key pairs	-	PCT	PCT	API return code	Encrypt and decrypt	Key Pair Generation
RSA Decrypt	2048-bit key	KAT	CAST	API return code	Decrypt	Module load
RSA Encrypt	2048-bit key	KAT	CAST	API return code	Encrypt	Module load
RSA Sign	SHA2-512, PKCS#1 V1.5, 2048-bit key	KAT	CAST	API return code	Sign	Module load
RSA Verify	SHA2-512, PKCS#1 V1.5, 2048-bit key	KAT	CAST	API return code	Verify	Module load
RSA signature key pairs	-	PCT	PCT	API return code	Sign and Verify	Key Pair Generation
SHA-2	SHA2-512	KAT	CAST	API return code	Generate	Module load
SHA-3	SHA3-512	KAT	CAST	API return code	Generate	Module load
SHAKE	SHAKE256	KAT	CAST	API return code	Generate	Module load
TLS 1.2 KDF	HMAC-SHA2-256	KAT	CAST	API return code	Derive	Module load
TLS 1.3 KDF	HMAC-SHA2-256	KAT	CAST	API return code	Derive	Module load
XTS Duplicate key	-	CT	Critical Function	API return code	Compare	On cipher initialization

Table 20: Conditional Self-Tests

10.2.1 Cryptographic Algorithm Self-tests

All Cryptographic Algorithm Self-tests (CASTs), are run prior to the Software Integrity Test. This ensures that the algorithms required by the Software Integrity Test have been tested prior to use by the Software Integrity Test.

All CASTs use a known answer test, KAT. For the KATs, the module includes a set of fixed inputs for each algorithm along with corresponding pre-calculated expected outputs. The cryptographic algorithm is run with the fixed inputs and the algorithm outputs are compared with the expected outputs. If there is any difference the algorithm self-test fails and the module cannot be used as this triggers a failure in the pre-operational self-testing.

10.2.2 Pair-wise Consistency Tests

The module performs a Pair-wise Consistency test each time the module generates a DSA/DH, ECDSA, EdDSA, ML-DSA, RSA, or RSA_CIPHER public/private key pair.

If the Pair-wise Consistency conditional test fails, the module throws a `SecurityException` and aborts the operation. A Pair-wise Consistency test failure does NOT disable the module.

10.3 Periodic Self-Test Information

The following self-tests can be invoked by the administrator as described in section 10.5 below. The module cannot invoke these tests automatically.

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
Integrity test	KAT	SW/FW Integrity	On demand	Manually

Table 21: Pre-Operational Periodic Information

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
AES CCM	KAT	CAST	On demand	Manually
AES CMAC	KAT	CAST	On demand	Manually
AES ECB	KAT	CAST	On demand	Manually
AES GCM	KAT	CAST	On demand	Manually
Counter DRBG	KAT	CAST	On demand	Manually
DH DSA key pairs	PCT	PCT	-	-
DSA Verify	KAT	CAST	On demand	Manually
Diffie-Hellman	KAT	CAST	On demand	Manually
ECDH	KAT	CAST	On demand	Manually
ECDSA Sign	KAT	CAST	On demand	Manually
ECDSA Verify	KAT	CAST	On demand	Manually
ECDSA key pairs	PCT	PCT	-	-
EdDSA Sign	KAT	CAST	On demand	Manually
EdDSA Verify	KAT	CAST	On demand	Manually
EdDSA key pairs	PCT	PCT	-	-
HMAC DRBG	KAT	CAST	On demand	Manually
HSS Verify	KAT	CAST	On demand	Manually
Hash DRBG	KAT	CAST	On demand	Manually
KDA OneStep	KAT	CAST	On demand	Manually
ML-DSA Key pairs	PCT	PCT	-	-
ML-DSA KeyGen	KAT	CAST	On demand	Manually
ML-DSA Sign	KAT	CAST	On demand	Manually
ML-DSA Verify	KAT	CAST	On demand	Manually
PBKDF	KAT	CAST	On demand	Manually

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
RSA Cipher key pairs	PCT	PCT	-	-
RSA Decrypt	KAT	CAST	On demand	Manually
RSA Encrypt	KAT	CAST	On demand	Manually
RSA Sign	KAT	CAST	On demand	Manually
RSA Verify	KAT	CAST	On demand	Manually
RSA signature key pairs	PCT	PCT	-	-
SHA-2	KAT	CAST	On demand	Manually
SHA-3	KAT	CAST	On demand	Manually
SHAKE	KAT	CAST	On demand	Manually
TLS 1.2 KDF	KAT	CAST	On demand	Manually
TLS 1.3 KDF	KAT	CAST	On demand	Manually
XTS Duplicate key	CT	Critical Function	-	-

Table 22: Conditional Periodic Information

10.4 Error States

Name	Description	Conditions	Recovery Method	Indicator
Error State	The module suspends all cryptographic operations and becomes non-responsive	POST or CAST Failure	Reload / Reboot Module	API return code
Soft error	The module outputs an error indicator then resumes normal operation	PCT failure	Automatic	API return code

Table 23: Error States

Upon entering an error states the module will throw a `SecurityException`. While in an error state, the module inhibits all data output.

10.5 Operator Initiation of Self-Tests

The module provides the `ModuleConfig.runSelfTests()` API to allow the operator to perform self-tests on-demand. This API runs the integrity tests after all CAST's have completed successfully.

11 Life-Cycle Assurance

11.1 Installation, Initialization, and Startup Procedures

The module is installed by adding *jcmFIPS-7.1.jar* to the application's class path.

The module is started by starting the application that references it. The module uses JDK services to perform the module startup when the application loads the module.

When loading the module, the `com.rsa.crypto.jcm.ModuleLoader.load`() method extracts arguments from the `com.rsa.cryptoj.jcm.JavaModuleProperties` class, which is created using the `com.rsa.cryptoj.jcm.CryptoJModulePropertiesFactory` class.

The following arguments are extracted:

- The module jar file
- The security level, specified as the constant `ModuleConfig.LEVEL_1`
- This should have a value of 1
- An optional `SelfTestEventListener` to use for logging power-up self-test events
- An optional `java.util.concurrent.ExecutorService` used for running the power-up self-tests
- An optional File for reading and writing the status of the algorithm power-up self-tests.

Using the specified `securityLevel` ensures that the module is loaded for use in a FIPS 140-3 Level 1 compliant manner.

Loading the module runs the integrity tests which must complete successfully before any cryptographic services are made available by the module. This ensures that the application has made no modification to the module as part of its development or installation. For more information about the Integrity Tests, see section 5 - Software Security.

The module starts in the approved Mode and Crypto Officer Role by default. Otherwise, to assume a role once the module is operational, construct a `FIPS140Context` object for the desired role using the `FIPS140Context.getFIPS140Context(int mode,int role)` method.

- The mode argument must be the value `FIPS140Context.MODE_FIPS140`.
- To retrieve the current mode of operation, call `FIPS140Context.getMode()`.
- The available role value is the constant `FIPS140Context.ROLE_CRYPTO_OFFICER`.
- No role authentication is required for operation of the module in Security Level 1 mode.

This object can then be used to perform cryptographic operations using the module.

The only permitted maintenance operation is to add a signature to the jar file by re-signing with an application certificate. Otherwise, application writers should not attempt to modify the module jar file as the module will refuse to load or perform cryptographic operations.

11.2 Administrator Guidance

For details of the administrative functions, security parameters, and logical interfaces available to the Crypto Officer, refer to Section 4.2 - Roles. The following details the requirements for compliant use of the module:

11.2.1 Algorithm usage

The modules usage of approved algorithms and security functions shall conform to the Algorithm Specific Information described in section 2.7 of this security policy.

11.2.3 Key Agreement

Obtaining domain parameters:

- For schemes using FFC, use one of the FFC safe-prime groups as defined in SP 800-56A Rev. 3 Appendix D.
- For schemes using ECC, use one of the approved curves as defined in SP 800-186.

Obtain a key pair from domain parameters:

For all schemes:

- Both parties must use validated parameters to generate a key pair.
- The module generates the key establishment key pair according to the required standards.
- Choose an Approved DRBG like HMAC DRBG to generate the key pair.
- Both parties validate the key pair.

The module provides the following APIs to explicitly validate the public and private keys according to SP 800-56A Rev.3:

```
com.rsa.crypto.PublicKey.isValid(SecureRandom random)
```

```
com.rsa.crypto.PrivateKey.isValid().
```

The module provides the APIs to explicitly validate the key pair according to the pairwise consistency requirements in SP 800-56A Rev. 3:

```
com.rsa.crypto.KeyPair.validate(SecureRandom random)
```

```
com.rsa.crypto.KeyPair.validate(AlgorithmParams params,
SecureRandom random)
```

If the key pair is generated using an approved method, then validation is assumed.

For schemes that use static key pairs, a public identifier must be:

- authoritatively associated with the key pair
- associated with the public key to allow any peer to recognize the key pair.

For schemes that use ephemeral keys, the key pair must be:

- used only for a single agreement transaction
- destroyed after use.

Receive the peer's public key:

For all schemes, the receiving party must validate the peer's public key.

For schemes that use static keys, the receiving party must have assurance of:

- the peer's ownership of the private key
- the identifier is bound to the public key.

Generate the Shared Secret.

For all schemes, the shared secret must be:

- used only as input to an approved KDF
- treated as a CSP and destroyed after use.

If the shared secret generation fails, then the party must destroy all intermediate values.

Generate and Confirm Secret Key Material:

For all schemes:

- Approved key-derivation method(s), including the format of `FixedInfo` as specified in SP 800-56A Rev. 3.
- When the shared secret is used as input to the KDF the outputs must be used as secret keys
- All key material must be generated before any of the keys are used
- If key generation fails, then the party must destroy all calculated values
- The shared secret and any key material is destroyed.

For schemes that use key confirmation:

- Both parties must use a common, approved MAC to generate confirmation values.
- The MAC key will be generated as one of the key material elements
- The input values for MAC tag generation must be formatted as per SP 800-56A Rev. 3
- The MAC key and tag lengths must satisfy the requirements of SP 800-56A Rev. 3
- The MAC key must be destroyed after use.
- If confirmation fails, then destroy all calculated values.

All key material is destroyed such that it cannot be used for any other purpose.

Approved key confirmation technique(s) as specified in SP 800-56A Rev. 3

11.2.4 Key Generation

When using an approved DRBG to generate keys, the security strength of the DRBG must be at least as great as the security strength of the key being generated. For details about the comparable security strengths of symmetric block ciphers and asymmetric key algorithms refer to Table 2 of NIST SP 800-57 Part 1 Rev. 5.

When generating key pairs using the `KeyPairGenerator` object, the `generate(boolean pairwiseConsistency)` method must not be invoked with an argument of false. Use of the no-argument `generate()` method is recommended.

11.2.5 Digital Signatures

Keys used for digital signature generation and verification shall not be used for any other purpose. The module generates keys with a particular purpose that is either signing or encryption. The same purpose must always be used for a given key when exported and loaded into the module again.

The length of an RSA key pair for digital signature generation must be greater than or equal to 2048 bits. For digital signature verification, the length must be greater than or equal to 1024 bits. RSA keys shall have a public exponent of an odd number, equal to or greater than 65537.

The SHA-1 digest is disallowed for the generation of digital signatures.

For RSASSA-PSS: If `nLen` is 1024 bits, and the output length of the approved hash function output block is 512 bits, then the length of the salt (`sLen`) shall be `0 <= sLen <= hLen - 2`. Otherwise, the length of the salt shall be `0 <= sLen <= hLen` where `hLen` is the length of the hash function output block (in bytes or octets).

11.2.6 Self-tests

The `com.rsa.cryptoj.fips140.katstatus.file` property has a default value of DISABLED. Changing this to any other value will cache the CAST results for the current platform and they will be skipped on subsequent module restarts. Since running the CASTs on every module startup is a requirement, this will put the module in a non-compliant state.

11.3 Non-Administrator Guidance

N/A. The module does not support a non-administrator role.

11.6 End of life

The module does not persistently store SSPs. No sanitization procedure exists or is necessary.

12 Mitigation of Other Attacks

12.1 Attack list

RSA, ECDSA, and DSA signature operations implement blinding by default, a reversible way of modifying the input data, so as to make the operation immune to timing attacks. Blinding has no effect on the algorithm other than to mitigate attacks on the algorithm. For more information, see Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.

RSA, ECDSA, and DSA blinding is implemented through blinding modes, for which the following options are available:

- Blinding mode off
- Blinding mode with no update, where the blinding value is squared for each operation.

This mitigation is enabled by default. For optimum security, it should not be disabled. For more information on configuring blinding, see the Dell BSAFE Crypto-J Developers Guide.

RSA signing operations implement a verification step after private key operations. This verification step is in place to prevent potential faults in optimized Chinese Remainder Theorem (CRT) implementations. It has no effect on the signature algorithm. For more information, see Modulus Fault Attacks Against RSA-CRT Signatures and On the Importance of Eliminating Errors in Cryptographic Computations.

This mitigation is enabled by default. For optimum security, it should not be disabled.

RSA PKCS #1 v1.5 encryption padding operations are implemented in constant time in order to make the operation immune to timing attacks. For more information, see Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1.

Time invariant comparisons are also used for HMAC and RSA verify operations. For this mitigation, constant time padding is built-in and cannot be disabled.

Acronyms

The following table lists the acronyms used with the module and their definitions:

Acronym	Definition
3DES	A symmetric encryption algorithm which uses either two or three DES keys. The two key variant of the algorithm provides 80 bits of security strength while the three key variant provides 112 bits of security strength.
AD	Authenticated Decryption. A function that decrypts purported ciphertext and verifies the authenticity and integrity of the data.
AE	Authenticated Encryption. A block cipher mode of operation which provides a means for the authenticated decryption function to verify the authenticity and integrity of the data.
AEAD	Authenticated Encryption with Associated Data.
AES	Advanced Encryption Standard. A fast block cipher with a 128-bit block, and keys of lengths 128, 192 and 256 bits. This will replace DES as the US symmetric encryption standard.
API	Application Programming Interface.
Attack	Either a successful or unsuccessful attempt at breaking part or all of a crypto-system. Attack types include an algebraic attack, birthday attack, brute force attack, chosen ciphertext attack, chosen plaintext attack, differential cryptanalysis, known plaintext attack, linear cryptanalysis, middleperson attack and timing attack.
BPS	BPS is a format preserving encryption mode. BPS stands for Brier, Peyrin and Stern, the inventors of this mode.
CAST	Cryptographic Algorithm Self-Tests. A test of all cryptographic functions of an approved cryptographic algorithm that must be performed prior to the first operational use of the cryptographic algorithm. A CAST may be a KAT, a comparison test or a fault-detection test.
CBC	Cipher Block Chaining. A mode of encryption in which each ciphertext depends upon all previous ciphertexts. Changing the IV alters the ciphertext produced by successive encryptions of an identical plaintext.
CFB	Cipher Feedback. A mode of encryption that produces a stream of ciphertext bits rather than a succession of blocks. In other respects, it has similar properties to the CBC mode of operation.
ChaCha20	A member of the ChaCha family of stream ciphers built on a pseudo-random function, based on add-rotate-XOR operations: 32-bit addition, bitwise addition (XOR) and rotation operations. ChaCha20 is standardized in RFC 7539.
ChaCha20/ Poly1305	A combination of the ChaCha20 and Poly1305 algorithms to provide an AEAD algorithm. This is standardized in RFC 7539.
CKG	Cryptographic Key Generation.
CMAC	Cipher-based Message Authentication Code. A block cipher-based MAC algorithm.
CMVP	Cryptographic Module Validation Program.
CRNG	Continuous Random Number Generation.
CSP	Critical Security Parameters.

CTR	Counter mode of encryption, which turns a block cipher into a stream cipher. It generates the next keystream block by encrypting successive values of a counter.
CTR DRBG	Counter mode Deterministic Random Bit Generator.
CTS	Cipher Text Stealing. A mode of encryption which enables block ciphers to be used to process data not evenly divisible into blocks, without the length of the ciphertext increasing.
Decryption	The conversion of encrypted data, ciphertext, into its original form. Generally, the reverse of encryption.
DES	Data Encryption Standard. A symmetric encryption algorithm which uses a 56-bit key with eight parity bits.
DH, Diffie-Hellman	The Diffie-Hellman asymmetric key exchange algorithm. There are many variants, but typically two entities exchange some public information (for example, public keys or random values) and combines them with their own private keys to generate a shared session key. As private keys are not transmitted, eavesdroppers are not privy to all of the information that composes the session key.
DPK	Data Protection Key.
DRBG	Deterministic Random Bit Generator.
DSA	Digital Signature Algorithm. An asymmetric algorithm for creating digital signatures.
EC	Elliptic Curve.
ECB	Electronic Code Book. A mode of encryption in which identical plaintexts are encrypted to identical ciphertexts, given the same key.
ECC	Elliptic Curve Cryptography.
ECDH	Elliptic Curve Diffie-Hellman.
ECCDH	Elliptic Curve Cryptography Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm.
ECIES	Elliptic Curve Integrated Encryption Scheme.
Encryption	The transformation of plaintext into an apparently less readable form (called ciphertext) through a mathematical process. The ciphertext may be read by anyone who has the key that decrypts (undoes the encryption) the ciphertext.
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standards.
FPE	Format Preserving Encryption.
GCM	Galois/Counter Mode. A mode of encryption combining the Counter mode of encryption with Galois field multiplication for authentication.
HKDF	HMAC-based Extract-and-Expand Key Derivation Function.
HMAC	Keyed-Hashing for Message Authentication Code.
HMAC DRBG	HMAC Deterministic Random Bit Generator.
IV	Initialization Vector. Used as a seed value for an encryption operation.

JCE	Java Cryptography Extension.
JVM	Java Virtual Machine.
KAT	Known Answer Test.
KDF	Key Derivation Function. Derives one or more secret keys from a secret value, such as a master key, using a pseudo-random function.
Key	A string of bits used in cryptography, allowing people to encrypt and decrypt data. Can be used to perform other mathematical operations as well. Given a cipher, a key determines the mapping of the plaintext to the ciphertext. Various types of keys include: distributed key, private key, public key, secret key, session key, shared key, subkey, symmetric key, and weak key.
Key wrapping	A method of encrypting key data for protection on untrusted storage devices or during transmission over an insecure channel.
KW	AES Key Wrap.
KWP	AES Key Wrap with Padding.
MAC	Message Authentication Code.
MD5	A secure hash algorithm created by Ron Rivest. MD5 hashes an arbitrary-length input into a 16-byte digest.
NDRNG	Non-Deterministic Random Number Generator.
NIST	National Institute of Standards and Technology. A division of the US Department of Commerce (formerly known as the NBS) which produces security and cryptography-related standards.
OFB	Output Feedback. A mode of encryption in which the cipher is decoupled from its ciphertext.
OS	Operating System.
PBE	Password-Based Encryption.
PBKDF	A method of password-based key derivation, originally defined in RFC2898 as PBKDF2, which applies a MAC algorithm to derive the key. In RFC2898 the PRF used is SHA-1. SP 800-132 approves PBKDF2 where the PRF may be any approved hash function. In this document PBKDF represents the expanded specification provided in SP 800-132.
Poly1305	A cryptographic MAC standardized in RFC 7539.
POST	Pre-Operational Self-Tests.
privacy	The state or quality of being secluded from the view or presence of others.
private key	The secret key in public key cryptography. Primarily used for decryption but also used for encryption with digital signatures.
PRNG	Pseudo-Random Number Generator.
RC2	Block cipher developed by Ron Rivest as an alternative to the DES. It has a block size of 64 bits and a variable key size. It is a legacy cipher and RC5 should be used in preference.
RC4	Symmetric algorithm designed by Ron Rivest using variable length keys (usually 40 bit or 128 bit).
RC5	Block cipher designed by Ron Rivest. It is parameterizable in its word size, key length and number of rounds. Typical use involves a block size of 64 bits, a key size of 128 bits and either 16 or 20 iterations of its round function.

RNG	Random Number Generator.
RSA	Public key (asymmetric) algorithm providing the ability to encrypt data and create and verify digital signatures. RSA stands for Rivest, Shamir, and Adleman, the developers of the RSA public key crypto-system.
SHA	Secure Hash Algorithm. An algorithm which creates a hash value for each possible input. SHA takes an arbitrary input which is hashed into a 160-bit digest.
SHA-1	A revision to SHA to correct a weakness. It produces 160-bit digests. SHA-1 takes an arbitrary input which is hashed into a 20-byte digest.
SHA-2	The NIST-mandated successor to SHA-1, to complement the Advanced Encryption Standard. It is a family of hash algorithms (SHA2-256, SHA2-384 and SHA2-512) which produce digests of 256, 384 and 512 bits respectively.
SHA-3	A family of hash algorithms which includes SHA3-224, SHA3-256, SHA3-384 and SHA3-512. It also includes the extendable-output functions SHAKE128 and SHAKE256. SHA-3 is an alternative to SHA-2, as no significant attacks on SHA-2 are currently known.
Shamir Secret Sharing	A form of secret sharing where a secret is divided into parts, and each participant is given a unique part. Some or all of the parts are needed to reconstruct the secret. This is also known as a (k, n) threshold scheme where any of the parts are sufficient to reconstruct the original secret.
TDES	Refer to 3DES.
TLS	Transport Layer Security.
Triple-DES	Refer to 3DES.
XTS	XEX-based Tweaked Codebook mode with ciphertext stealing. A mode of encryption used with AES.