BSAFE™ Java Crypto Module 6.3 Security Policy Level 1

This document is a non-proprietary security policy for the BSAFE Java Crypto Module 6.3 (BSAFE Crypto Module) security software from Dell Australia Pty Limited, BSAFE Product Team.

This document may be freely reproduced and distributed whole and intact including the copyright notice.

Contents:

Preface

Federal Information Processing Standards Publication 140-2 - Security Requirements for Cryptographic Modules (FIPS 140-2) details the U.S. Government requirements for cryptographic modules. More information about the FIPS 140-2 standard and validation program is available on the NIST website.

With the exception of the non-proprietary *BSAFE Java Crypto Module Security Policy* documents, the FIPS 140-2 Security Level 1 validation submission documentation is proprietary to Dell Australia Pty Limited, BSAFE Product Team. and is releasable only under appropriate non-disclosure agreements. For access to the documentation, contact Dell Customer Support.

This security policy describes how the BSAFE Crypto Module meets the overall Security Level 1 security requirements of FIPS 140-2, and how to securely operate it.

This document deals only with operations and capabilities of the BSAFE Crypto Module in the technical terms of a FIPS 140-2 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 FIPS 140-2 validated Cryptographic Module for Overall Security Level 1 with Level 3 Design Assurance.

The BSAFE Crypto Module is also referred to as:

- The Cryptographic Module
- The Java Crypto Module (JCM)
- The module

1 The Cryptographic Module

This section provides an overview of the module, and contains the following topics:

1.1 Module Characteristics

The JCM is classified as a FIPS 140-2 multi-chip standalone module. As such, the JCM is tested on particular operating systems and computer platforms. The cryptographic boundary includes the JCM running on selected platforms that are running selected operating systems.

The JCM is validated for FIPS 140-2 Security Level 1 requirements.

The following table lists the certification levels sought for the JCM for each section of the FIPS 140-2 specification:

Table 1 Certification Levels

Section of the FIPS 140-2 Specification	Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	3
Mitigation of Other Attacks	1
Overall	1

The JCM is packaged in a Java Archive (JAR) file called `jcmFIPS-6.3.jar` containing all the code for the module.

1.1.1 Laboratory Validated Operating Environments

For FIPS 140-2 validation, the JCM is tested by an accredited FIPS 140-2 testing laboratory on the following operating environments:

Table 2 Tested Operating Environments

#	Operating System	Hardware Platform	Processor	PAA[1]
	SUSE® Software Solutions®			
1	SUSE Linux Enterprise Server 15 SP3 (64-bit) with OpenJDK 11	Dell PowerEdge R6525	AMD EPYC™ 7513	No
	Microsoft®			
2	Windows Server 2019 (64-bit) with Oracle® JRE 8	Dell PowerEdge R6525	AMD EPYC™ 7513	No

[1]Processor Algorithm Acceleration

1.1.2 Affirmation of Compliance for other Operating Environments

Affirmation of compliance is defined in Section G.5, “Maintaining Validation Compliance of Software or Firmware Cryptographic Modules,” in Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program. Compliance is maintained in all operational environments for which the binary executable remains unchanged. Specifically, Dell Australia Pty Limited, BSAFE Product Team affirms compliance for the following operational environments:

Table 3 Vendor Affirmed Operational Environments

#	Operating System	Hardware Platform
Canonical®		
1	Ubuntu® 16.04 (x86_64) with OpenJDK JDK 8 (64-bit)	Intel x86_64 (64-bit)
CentOS™ Project		
2	CentOS 7.9 (x86_64) with OpenJDK JDK 8 (64-bit)	Intel x86_64 (64-bit)
Dell™		
3	PowerProtect™ Data Domain™ OS (x86_64) with Oracle JDK 8 (64-bit)	Intel x86_64 (64-bit)
FreeBSD® Foundation		
4	FreeBSD 12 (x86_64) with OpenJDK JDK 8 (64-bit)	Intel x86_64 (64-bit

Table 3 Vendor Affirmed Operational Environments (continued)

#	Operating System		Hardware Platform
HPE			
5	HP-UX 11.31 with HP JDK 8 (64-bit)		Itanium 2®
IBM			
6	AIX® 7.2 with IBM JDK 8 (64-bit)		PowerPC® (64-bit)
Microsoft			
	Windows® 10 Enterprise (x86_64)		
7		with Oracle JDK 11 (64-bit)	Intel x86_64 (64-bit)
8		with Oracle JDK 8 (64-bit)	
	Windows Server 2019 (x86_64)		
9		with Oracle JDK 11 (64-bit)	Intel x86_64 (64-bit)
10		with Oracle JDK 8 (64-bit)	
Oracle			
	Solaris® 11.4		
11		with Oracle JDK 11 (64-bit)	SPARC® v9
12		with Oracle JDK 8 (64-bit)	
Red Hat®			
	Enterprise Linux 8.6 (x86_64)		
13		with Oracle JDK 11 (64-bit)	Intel x86_64 (64-bit)
14		with Oracle JDK 8 (64-bit)	
	Enterprise Linux 7.9 (x86_64)		
15		with Oracle JDK 11 (64-bit)	Intel x86_64 (64-bit)
16		with Oracle JDK 8 (64-bit)	
SUSE Software Solutions®			
	SUSE® Linux Enterprise Server 15 SP4 (x86_64)		Intel x86_64 (64-bit)
17		with OpenJDK JDK 11 (64-bit)	
	SUSE Linux Enterprise Server 15 SP3 (x86_64)		

Table 3 Vendor Affirmed Operational Environments (continued)

#	Operating System		Hardware Platform
18		with OpenJDK JDK 11 (64-bit)	Intel x86_64 (64-bit)
19		with OpenJDK JDK 8 (64-bit)	
	SUSE Linux Enterprise Server 12 SP5		
20		with Oracle JDK 11 (64-bit)	Intel x86_64 (64-bit)
21		with Oracle JDK 8 (64-bit)	
22		with IBM JDK 8 (64-bit)	

1.2 Module Interfaces

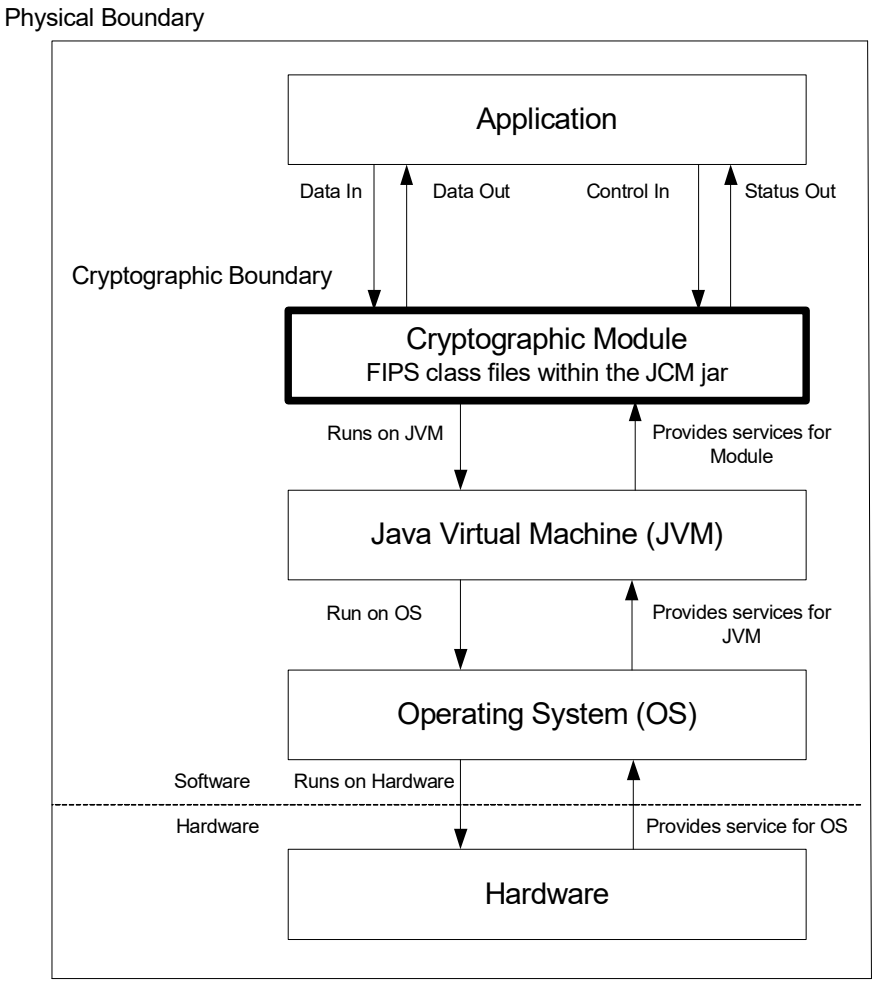
As a multi-chip standalone module, the physical interface to the JCM consists of a keyboard, mouse, monitor, serial ports and network adapters.

The underlying logical interface to the module is the API, documented in the relevant API *Javadoc*. The module provides the following four logical interfaces that have been designed within the module where input and output are indicated from the perspective of the module:

- Control In - the invocation of all methods, the function and API names
- Data In - input arguments to all constructors and methods specifying input parameters
- Data Out - modified input arguments, those passed by reference, and return values for all constructors and methods modifying input arguments and returning values
- Status Out - information returned by the methods and any exceptions thrown by constructors and methods.

This is shown in the following diagram.

Figure 1 JCM Logical Diagram



1.3 Roles and Services

The JCM is designed to meet all FIPS 140-2 Level 1 requirements, implementing both a Crypto Officer role and a Crypto User role. As allowed by FIPS 140-2, the JCM does not require user identification or authentication for these roles.

1.3.1 Crypto Officer Role

The Crypto Officer is responsible for installing and loading the module. Once the module has been installed and is operational, an operator can assume 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`.

The Services section provides a list of services available to the Crypto Officer Role.

1.3.2 Crypto User Role

An operator can assume the Crypto User 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_USER`.

The Services section provides a list of services available to the Crypto User Role.

1.3.3 Services

The JCM provides services which are available for **both** FIPS 140-2 and non-FIPS 140-2 usage. For a list of FIPS 140-2 approved and FIPS 140-2 allowed algorithms, see FIPS 140-2 Approved Algorithms.

The following table lists the un-authenticated services provided by the JCM which may be used by either Role, in either the FIPS or non-FIPS mode, in terms of the module interface. For each interface, lists of algorithms that are allowed and not allowed when operating the module in a FIPS 140-2 compliant way are specified.

Table 4 Services Available to the Crypto User and Crypto Officer Roles

Services Available to the Crypto User and Crypto Officer Roles		
Encryption/Decryption:		

`SymmCipher`	`clearSensitiveData`
	`clone`
	`doFinal`
	`getAlg`
	`getAlgorithmParams`
	`getBlockSize`
	`getCryptoModule`
		`getFeedbackSize`
		`getMaxInputLen`
		`getOutputSize`
		`init`
		`isIVRequired`
		`reInit`
		`update`

	Algorithms approved for FIPS 140-2 usage	
	AES (CBC, CCM,CFB128, CTR, ECB, GCM, OFB, XTS) AES (CBC_CS1, CBC_CS2, CBC_CS3)	
	Algorithms not allowed for FIPS 140-2 usage	
	AES BPS Triple-DES (CBC, CFB, ECB, OFB) Triple-DES (CBC_CS1, CBC_CS2, CBC_CS3) ChaCha20 ChaCha20/Poly1305 DES DESX RC2® RC4® RC5® PBE (PKCS #12, PKCS #5, SSLCPBE)	

Table 4 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles		
Encryption/Decryption: (continued)		

`Cipher`	`clearSensitiveData`
	`clone`
	`doFinal`
	`getAlg`
	`getAlgorithmParams`
	`getBlockSize`
		`getCryptoModule`
		`getMaxInputLen`
		`getOutputSize`
		`init`
		`reInit`
		`update`

	Algorithms approved for FIPS 140-2 usage	
	SP 800-38F KW (AE, AD, provides between 128 and 256 bits of encryption strength) SP 800-38F KWP (AE, AD, provides between 128 and 256 bits of encryption strength)	
	Algorithms not allowed for FIPS 140-2 usage	
	ECIES RSA OAEP RSA-KEM-KWS	
Signature Generation/Verification:		

`Signature`	`clearSensitiveData`
	`clone`
	`getAlg`
	`getCryptoModule`
	`getSignatureSize`
	`initSign`
		`initVerify`
		`reInit`
		`sign`
		`update`
		`verify`

	Algorithms approved for FIPS 140-2 usage	
	RSA X9.31, PKCS #1 V.1.5, RSASSA-PSS DSA ECDSA	
	Algorithms not allowed for FIPS 140-2 usage	
	None	

Table 4 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles		
MAC Generation/Verification:		

MAC	`clearSensitiveData`
	`clone`
	`getAlg`
	`getCryptoModule`
	`getMacLength`
		`init`
		`mac`
		`reset`
		`update`
		`verify`

	Algorithms approved for FIPS 140-2 usage	
	AES (CMAC) HMAC (when used with an approved Message Digest algorithm)	
	Algorithms not allowed for FIPS 140-2 usage	
	HMAC-MD5 PBHMAC (PKCS #12, PKIX) Poly1305	
Digest Generation:		

MessageDigest	`clearSensitiveData`
	`clone`
	`digest`
	`getAlg`
		`getCryptoModule`
		`getDigestSize`
		`reset`
		`update`

	Algorithms approved for FIPS 140-2 usage	
	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512 SHA-512/224 SHA-512/256	SHA3-224 SHA3-256 SHA3-384 SHA3-512 SHAKE128 SHAKE256
	Algorithms not allowed for FIPS 140-2 usage	
	MD2 MD5 (Allowed in FIPS mode only for use in TLS) RIPEMD160	

Table 4 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles		
Parameters:		
`AlgInputParams`	`clone` `get`	`getCryptoModule` `set`
`AlgorithmParams`	`getCryptoModule`	

`DHParams`	`getCounter`
	`getCryptoModule`
	`getG`
	`getJ`
		`getMaxExponentLen`
		`getP`
		`getQ`
		`getSeed`

`DomainParams`	`getCryptoModule`	

`DSAParams`	`getCounter`
	`getCryptoModule`
	`getDigestAlg`
	`getG`
		`getP`
		`getQ`
		`getSeed`

`ECParams`	`getA`
	`getB`
	`getBase`
	`getBaseDigest`
	`getBaseSeed`
	`getCofactor`
	`getCryptoModule`
	`getCurveName`
		`getDigest`
		`getFieldMidTerms`
		`getFieldPrime`
		`getFieldSize`
		`getFieldType`
		`getOrder`
		`getSeed`
		`getVersion`

`ECPoint`	`clearSensitiveData` `getEncoded`	`getX` `getY`
`PQGParams`	`getCryptoModule` `getG`	`getP` `getQ`
Parameter Generation:		

`AlgParamGenerator`	`generate`
	`getCryptoModule`
	`initGen`
		`initVerify`
		`verify`

	Algorithms approved for FIPS 140-2 usage	
	DSA KAS-SSC (KAS-FFC-SSC)	
	Algorithms not allowed for FIPS 140-2 usage	
	None	

Table 4 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles		
Key Establishment:		

`KeyAgreement`	`clearSensitiveData`
	`clone`
	`doPhase`
	`getAlg`
		`getCryptoModule`
		`getSecret`
		`init`

	Algorithms approved for FIPS 140-2 usage	
	KAS-SSC: KAS-FFC-SSC, and KAS-ECC-SSC (provides between 112 bits and 256 bits of encryption strength)	
	Algorithms not allowed for FIPS 140-2 usage	
	None	
`KeyConfirmation`	`clearSensitiveData` `computeMacTag`	`verifyMacTag`
	Algorithms approved for FIPS 140-2 usage	
	CVL (KAS-KC) (SP 800-56Ar3 / SP 800-56Br2)	
	Algorithms not allowed for FIPS 140-2 usage	
	None	
Key Generation:		
`KeyGenerator`[1]	`clearSensitiveData` `generate`	`getCryptoModule` `initialize`
	Algorithms approved for FIPS 140-2 usage	
	AES	
	Algorithms not allowed for FIPS 140-2 usage	
	DES DESX Shamir’s Secret Sharing Triple-DES	RC2 RC4 RC5

Table 4 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles		
Key Generation: (continued)		
KeyPairGenerator	clearSensitiveData clone generate	getCryptoModule initialize
	Algorithms approved for FIPS 140-2 usage	
	EC (ECDSA) DSA	KAS-SSC (KAS-ECC-SSC and KAS-FFC-SSC) RSA
	Algorithms not allowed for FIPS 140-2 usage	
	RSA Keypair Generation MultiPrime	
Keys:		
DHPrivateKey	clearSensitiveData clone getAlg getCryptoModule	getParams getX isValid
DHPublicKey	clearSensitiveData clone getAlg getCryptoModule	getParams getY isValid
DSAPrivateKey	clearSensitiveData clone getAlg getCryptoModule	getParams getX isValid
DSAPublicKey	clearSensitiveData clone getAlg getCryptoModule	getParams getY isValid
ECPrivateKey	clearSensitiveData clone getAlg getCryptoModule	getD getParams isValid
ECPublicKey	clearSensitiveData clone getAlg getCryptoModule	getParams getPublicPoint isValid
Key	clearSensitiveData clone	getAlg getCryptoModule

Table 4 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles		
Keys: (continued)		
KeyBuilder	newDHParams newDHPrivateKey newDHPublicKey newDSAParams newDSAPrivateKey newDSAPublicKey newECParams newECPrivateKey	newECPublicKey newPasswordKey newPQGParams newRSAPrivateKey newRSAPublicKey newSecretKey recoverShamirSecretKey
KeyPair	clearSensitiveData clone getAlgorithm	getPrivate getPublic
PasswordKey	clearSensitiveData clone getAlg	getCryptoModule getKeyData getPassword
PrivateKey	clearSensitiveData clone getAlg	getCryptoModule isValid
PublicKey	clearSensitiveData clone getAlg	getCryptoModule isValid
RSAPrivateKey	clearSensitiveData clone getAlg getCoeff getCryptoModule getD getE getExpP	getExpQ getN getOtherMultiPrimeInfo getP getQ hasCRTInfo isMultiprime isValid
RSAPublicKey	clearSensitiveData clone getAlg getCryptoModule	getE getN isValid
SecretKey	clearSensitiveData getAlg	getCryptoModule getKeyData
SharedSecretKey	clearSensitiveData clone getAlg	getCryptoModule getKeyData getSharedParams

Table 4 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles		
Key Derivation:		
KDF	clearSensitiveData clone	generate getCryptoModule
	Algorithms approved for FIPS 140-2 usage	
	CVL (KDFTLS12, for use with TLS 1.2) One-step KDF (SP 800-56C Rev. 1, KDA) KBKDF (SP 800-108) PBKDF2 (Approved for key storage)	
	Algorithms not allowed for FIPS 140-2 usage	
	KDFTLS10 (For use with TLS 1.0 and 1.1) PKCS #5 KDF PKCS #12 KDF scrypt	
Random Number Generation:		
SecureRandom	autoseed clearSensitiveData getCryptoModule newInstance nextBytes	selfTest setAlgorithmParams setOperationalParameters setSeed
	Algorithms approved for FIPS 140-2 usage	
	CTR DRBG Hash DRBG	HMAC DRBG
	Algorithms not allowed for FIPS 140-2 usage	
	FIPS 186-2 PRNG (Change Notice General)	
Other Services:		
BigNum	getBitLength	toOctetString
CryptoModule	getDeviceType getKeyBuilder getModuleOperations newAlgInputParams newAlgParamGenerator newAsymmetricCipher newKDF newKeyAgreement	newKeyGenerator newKeyPairGenerator newKeyWrapCipher newMAC newMessageDigest newSecureRandom newSignature newSymmetricCipher
JCMCloneable	clone	

Table 4 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles		
Other Services: (continued)		
ModuleConfig	getEntropySource getFIPS140Context getSecurityLevel getVersionDouble getVersionInfo getVersionString	initFIPS140RolePINs isFIPS140Compliant newCryptoModule runSelfTests setEntropySource
ModuleLoader	load	
ModuleOperations	perform	
PasswordKey	clearSensitiveData clone getAlg	getCryptoModule getKeyData getPassword
SelfTestEvent	getTestId	getTestName
SelfTestEventListener	failed finished	passed started
SensitiveData	clearSensitiveData	

[1]The key generator service generates keys, by using a specified SecureRandom, which can be used in algorithms such as AES (in Approved mode) and Triple-DES (in non-Approved mode).

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

1.4 Cryptographic Key Management

Cryptographic key management is concerned with generating and storing keys, protecting keys during use, zeroizing keys when they are no longer required and managing access to keys.

1.4.1 Key Generation

The module supports the generation of the DSA, KAS-FFC-SSC, RSA, and ECC public and private keys. In the FIPS-Approved mode, RSA keys can only be generated using the Approved 186-4 RSA key generation method.

The module uses the AES Counter-mode DRBG (AES-128-CTR DRBG) as the default pseudo-random number generator (PRNG) for generating asymmetric and symmetric keys.

1.4.2 Key Assurance

The module supports validity assurance of asymmetric keys. Methods are available to test the validity of:

- ECC keys, and DSA keys and domain parameters, against FIPS 186-4

- ECC keys, and KAS-FFC-SSC keys and domain parameters, against SP 800-56A Rev. 3
- RSA keys against FIPS 186-4 or SP 800-56B Rev. 2.

1.4.3 Key Protection

All key data resides in internally allocated data structures and can only be output using the JCM API. The operating system and the JRE safeguards memory and process space from unauthorized access.

1.4.4 Key Zeroization

The module stores all its keys and Critical Security Parameters (CSPs) in volatile memory. Users can ensure CSPs are properly zeroized by making use of the `<object>.clearSensitiveData()` method. All of the module's keys and CSPs are zeroizable. The module operator must zeroize all keys before switching from an approved FIPS 140-2 mode to non-FIPS 140-2 approved mode.
For more information about clearing CSPs, see the relevant API *Javadoc*.

1.4.5 Key Storage

The JCM does not provide long-term cryptographic key storage. Storage of keys is the responsibility of the JCM user. The Crypto User and Crypto Officer roles have equal and complete access to all keys and CSPs.

The following table shows how the storage of keys and CSPs are handled:

Table 5 Key and CSP Storage

Item	Storage
AES keys	In volatile memory only (plaintext)
CMAC keys	In volatile memory only (plaintext)
CTR DRBG Entropy	In volatile memory only (plaintext)
CTR DRBG init_seed	In volatile memory only (plaintext)
CTR DRBG Key	In volatile memory only (plaintext)
CTR DRBG V Value	In volatile memory only (plaintext)
DSA private key/public key	In volatile memory only (plaintext)
ECC private keys/public key	In volatile memory only (plaintext)
Hash DRBG C	In volatile memory only (plaintext)
Hash DRBG Entropy	In volatile memory only (plaintext)
Hash DRBG init_seed	In volatile memory only (plaintext)
Hash DRBG V Value	In volatile memory only (plaintext)

Table 5 Key and CSP Storage (continued)

Item	Storage
HMAC DRBG Entropy	In volatile memory only (plaintext)
HMAC DRBG init_seed	In volatile memory only (plaintext)
HMAC DRBG Key	In volatile memory only (plaintext)
HMAC DRBG V Value	In volatile memory only (plaintext)
HMAC with SHA-1, SHA-2 and SHA-3 keys	In volatile memory only (plaintext)
KAS-ECC-SSC Shared Secret	In volatile memory only (plaintext)
KAS-FFC-SSC private key/public key	In volatile memory only (plaintext)
KAS-FFC-SSC Shared Secret	In volatile memory only (plaintext)
One Step KDF secret	In volatile memory only (plaintext)
RSA private key/public key	In volatile memory only (plaintext)
SP 800-108 KDF secret	In volatile memory only (plaintext)
Katstatus	In volatile memory and on disk (protected with HMAC SHA256)

1.4.6 Key Access

An authorized operator of the module has access to all key data created during JCM operation. The User and Officer roles have equal and complete access to all keys.

The following table lists the different services provided by the module with the type of access to keys or CSPs:

Table 6 Key and CSP Access

Service	Key or CSP	Type of Access
Asymmetric encryption and decryption	Asymmetric keys (RSA)	Read/Execute
Symmetric encryption and decryption	Symmetric keys (AES)	Read/Execute
Digital signature and verification	Asymmetric keys (DSA, ECDSA, RSA)	Read/Execute
Message digest	None	N/A
MAC	HMAC keys CMAC keys	Read/Execute
Random number generation	CTR DRBG entropy, V, key, init_seed Hash DRBG entropy, V, C, init_seed HMAC DRBG entropy, V, key, init_seed	Read/Write/Execute

Table 6 Key and CSP Access (continued)

Service	Key or CSP	Type of Access
Key assurance	Asymmetric keys (DSA, ECC, KAS-FFC-SSC and RSA)[1]	Read
Key derivation	SP 800-108 KDF secret One-step KDF secret	Read/Execute
Key establishment primitives	Asymmetric keys (KAS-ECC-SSC, KAS-FFC-SSC)	Read/Execute
Key generation	Symmetric keys (AES) Asymmetric keys (DSA, ECDSA, KAS-ECC-SSC, KAS-FFC-SSC, RSA) MAC keys (HMAC, CMAC)	Write
Self-test	Hard-coded keys, (AES, RSA, DSA, ECDSA, HMAC, CMAC, SP 800-108 KDF) Hard-coded entropy, strength, and seed (HMAC DRBG, HASH DRBG, CTR DRBG)	Read/Execute
Show status	None	N/A
Zeroization	All	Read/Write

[1]For details of asymmetric key assurance, see section 1.4.2 Key Assurance.

1.5 Cryptographic Algorithms

The JCM offers a wide range of cryptographic algorithms. This section describes the algorithms that can be used when operating the module in a FIPS 140-2 compliant manner.

1.5.1 FIPS 140-2-approved Algorithms

The following table lists the BSAFE Crypto Module FIPS 140-2 Approved algorithms, with the appropriate standards and CAVP validation certificate:

Table 7 FIPS 140-2 Approved Algorithms

AVP Cert	Algorithm and Standard	Mode/Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A3218	AES SP 800-38A	CBC, CFB128, CTR, ECB, OFB,	128, 192, 256 bit key sizes	Symmetric encryption/decryption
A3218	AES SP 800-38B	CMAC-AES	128, 192, 256 bit key sizes	Message authentication
A3218	AES SP 800-38C	CCM	128, 192, 256 bit key sizes	Symmetric encryption/decryption
A3218	AES SP 800-38D	GCM	128, 192, 256 bit key sizes	Symmetric encryption/decryption
A3218	AES SP 800-38E	XTS	128, 256 bit key sizes	Symmetric encryption/decryption
A3218	KTS[1] SP 800-38F	Key Wrap and Key Wrap with Padding	128, 192, 256 bit key sizes	Key Wrapping / Unwrapping
A3218	CVL (KAS ECC CDH Component) SP 800-56A Rev.3	Full Public Key Validation, Key Pair Generation, Partial Public Key Validation	224 to 521 bit key sizes	Key agreement primitive
A3218	KAS-SSC (KAS-ECC-SSC) SP 800-56A Rev.3	ephemeralUnified, staticUnified with functions: keyPairGen, partialVal, fullVal	224 to 521 bit key sizes[2]	Key agreement primitive
A3218	KAS-SSC (KAS-FFC-SSC) SP 800-56A Rev. 3	dhEphem, dhOneFlow, dhStatic	2048 to 8192 bit key sizes[3] The bit length of the private key, N, is calculated as len(q), where q is the sub-prime of the selected Safe Prime domain parameters.	Key agreement primitive

AVP Cert	Algorithm and Standard	Mode/Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A3218	CVL (KAS KC) SP 800-56A Rev. 3/ SP 800-56B Rev. 2	HMAC SHA-1, HMAC SHA2-224, HMAC SHA2-256, HMAC SHA2-384, HMAC SHA2-512, HMAC SHA2-512/224 , HMAC SHA2-512/256 , HMAC SHA2-224, HMAC SHA3-224, HMAC SHA3-256, HMAC SHA3-384, HMAC SHA3-512	128 bit key size	Key agreement primitive
A3218	Safe Primes SP 800-56A Rev. 3	Key generation, Key verification	2048 to 8192 bit key sizes	Key agreement primitive
A3218	KAS-RSA-SSC[4] SP 800-56B Rev. 2	KAS1	2048 to 8192 bit key sizes	Key agreement primitive
A3218	CVL (RSADP) SP 800-56B Rev. 2	decryptionPrimitive	2048 bit key size	Key transport
A3218	KDA SP 800-56C Rev. 1	OneStep with digests: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512	112 to 256 bit key strengths	Key derivation
A3218	DRBG SP 800-90A Rev. 1	CTR_DRBG with AES-CTR	128, 192, 256 bit key strengths	Random bit generation
A3218	DRBG SP 800-90A Rev. 1	HASH_DRBG with Hash SHA1, Hash SHA2-224, Hash SHA2-256, Hash SHA2-384, Hash SHA2-512, Hash SHA2-512/224, Hash SHA2-512/256	128, 192, 256 bit key strengths	Random bit generation

AVP Cert	Algorithm and Standard	Mode/Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A3218	DRBG SP 800-90A Rev. 1	HMAC_DRBG with HMAC SHA1, HMAC SHA2-224, HMAC SHA2-256, HMAC SHA2-384, HMAC SHA2-512, HMAC SHA2-512/224, HMAC SHA2-512/256	128, 192, 256 bit key strengths	Random bit generation
A3218	KBKDF SP 800-108	Feedback with HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512, HMAC-SHA2-512/224, HMAC-SHA2-512/256	112 to 256 bit key strengths	Key derivation
A3218	PBKDF[5] SP 800-132		112 to 256 bit key strengths	Key derivation
A3218	CVL TLS 1.2 KDF[6] RFC 7627		128 to 256 bit key strengths	Key derivation
A3218	SHS FIPS 180-4	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256		Secure hashing
A3218	RSA FIPS 186-4	Key Generation	2048, 3072, 4096 bit key sizes	Digital signatures
A3218	RSA FIPS 186-4	Signature Generation with ANSI X9.31, PKCS #1 V.1.5, PKCSPSS	2048, 3072, 4096 bit key sizes	Digital signatures
A3218	RSA FIPS 186-4	Signature Verification with ANSI X9.31, PKCS #1 V.1.5, PKCSPSS	1024, 2048, 3072, 4096 bit key sizes	Digital signatures
A3218	DSA FIPS 186-4	Parameter Generation, Key Generation, Signature Generation	2048, 3072 bit key sizes	Digital signatures
A3218	DSA FIPS 186-4	Parameter Verification, Signature Verification	1024, 2048, 3072 bit key sizes	Digital signatures

AVP Cert	Algorithm and Standard	Mode/Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A3218	ECDSA FIPS 186-4	Key Generation, Key Verification, Signature Generation, Signature Verification	224 to 571 bit key sizes	Digital signatures
VA[7]	Cryptographic Key Generation (CKG)[8]	SP 800-133 Rev. 2		Key Generation
A3218	HMAC FIPS 198-1	HMAC SHA-1, HMAC SHA2-224, HMAC SHA2-256, HMAC SHA2-384, HMAC SHA2-512, HMAC SHA2-512/224 , HMAC SHA2-512/256 , HMAC SHA2-224, HMAC SHA3-224, HMAC SHA3-256, HMAC SHA3-384, HMAC SHA3-512	112 to 256 bit key strengths	Message authentication
A3218	SHA-3 FIPS 202	SHA3-224, SHA3-256, SHA3-384, SHA3-512		Secure hashing
A3218	SHA-3 FIPS 202	SHAKE-128, SHAKE-256		Secure hashing

[1]AES (key wrapping; SSP establishment methodology provides between 112 and 256 bits of encryption strength).

[2]KAS-ECC-SSC: Key establishment methodology provides between 112 and 256 bits of encryption strength.

[3]KAS-FFC-SSC: Key establishment methodology provides between 112 and 200 bits of encryption strength.

[4]KAS-RSA-SSC is also referred to as KAS-IFC-SSC.

[5]Password-based key derivation function 2 (PBKDF2). As defined in NIST Special Publication 800-132, PBKDF2 can be used in FIPS 140-2 mode when used with FIPS 140-2 Approved symmetric key and message digest algorithms. For more information, see Crypto Officer Guidance.

[6]The TLS 1.2 KDF, documented in SP 800-135, is allowed only when the conditions detailed in the Crypto User Guidance are satisfied. JCM does not implement the full TLS protocol and only the implemented components have been tested by the CAVP and CMVP.

[7]Vendor Affirmed.

[8]The module supports cryptographic key generation as described in SP 800-133 Rev. 2 section 4, where V is a constant string of binary zeroes. The module also supports symmetric key generation as described in sections 6.1 and 6.2.

1.5.2 Non-FIPS 140-2 and Not Allowed in the Approved Mode Algorithms

The following table lists all other available algorithms in the JCM that are not allowed for FIPS 140-2 usage. These algorithms must not be used when operating the module in a FIPS 140-2 compliant way.

Table 8 BSAFE Crypto Module Non-FIPS 140-2 and Not Allowed in the Approved Mode Algorithms

Algorithm / Function	**Use / Function**
AES in BPS mode for FPE	Symmetric encryption / decryption
AES in CBC-CS1, CBC-CS2 and CBC-CS3 mode for CTS	Symmetric encryption / decryption
ChaCha20	Symmetric encryption / decryption
ChaCha20/Poly1305	Symmetric encryption / decryption
DES	Symmetric encryption / decryption
DESX	Symmetric encryption / decryption
ECIES	Asymmetric encryption / decryption
FIPS 186-2 PRNG (Change Notice General)	Random bit generation
HMAC-MD5	Message authentication
KDFTLS10	Key Derivation (For use with TLS 1.0 and 1.1)
MD2	Secure hashing
MD5	Secure hashing
PBE (PKCS #12, PKCS #5, SSLCPBE)	Symmetric encryption / decryption
PBHMAC (PKCS #12, PKIX)	Message authentication
Poly1305	Message authentication
RC2	Symmetric encryption / decryption
RC4	Symmetric encryption / decryption
RC5	Symmetric encryption / decryption
RIPEMD160	Secure hashing
RSA-KEM-KWS	Asymmetric encryption / decryption
RSA-OAEP	Asymmetric encryption / decryption

Table 8 BSAFE Crypto Module Non-FIPS 140-2 and Not Allowed in the Approved Mode Algorithms

Algorithm / Function	**Use / Function**
scrypt	Key Derivation
Shamir's Secret Sharing	Key Generation
TDES in CBC, CFB64, ECB, OFB modes and CBC_CS1, CBC_CS2 or CBC_CS3 mode for CTS	Symmetric encryption / decryption

1.6 Self-tests

The module performs power-up and conditional self-tests to ensure proper operation.

If the power-up self-test fails, the module is disabled and throws a `SecurityException`. The module cannot be used within the current JVM.
If the conditional self-test fails, the module throws a `SecurityException` and aborts the operation. A conditional self-test failure does NOT disable the module.

1.6.1 Power-up Self-tests

Power-up self-tests are executed automatically when the module is loaded into memory. The power-up self-tests include the FIPS 140-2 required Software Integrity Test and a set of Cryptographic Algorithms tests. The Software Integrity Test is comprised of an HMAC-SHA1 verification of the files listed in *fips140/module.files*.

The following Cryptographic Algorithm tests are implemented in the module:

- AES/ECB Decrypt KAT
- AES/ECB Encrypt KAT
- AES/CCM Decrypt KAT
- AES/CCM Encrypt KAT
- AES/GCM Decrypt KAT
- AES/GCM Encrypt KAT
- CMAC KAT
- CTR DRBG KAT
- DSA Signature KAT
- DSA Verify KAT
- ECDSA Signature KAT
- ECDSA Verify KAT
- Hash DRBG KAT
- HMAC DRBG KAT
- KAS-ECC-SSC Primitive Z KAT

- KAS-FFC-SSC Primitive Z KAT
- KAS-RSA-SSC Primitive Decryption KAT
- KAS-RSA-SSC Primitive Encryption KAT
- KDFTLS12 SHA-256 KAT
- OneStep KDF KAT
- PBKDF2 KAT
- RSA Signature KAT
- RSA Verify KAT
- SHA3-512 KAT
- SHA-512 KAT
- SHAKE256 KAT
- SP 800-108 KDF KAT

By default, all Cryptographic Algorithm tests are run at power-up. However, if configured to do so, the module will run all of the power-up self-tests when first loaded in an operational environment, and run only the Software Integrity Test on subsequent restarts.

1.6.2 Conditional Self-tests

The module performs two conditional self-tests:

- Pair-wise Consistency Tests each time the module generates a DSA, KAS-FFC-SSC, RSA or EC public/private key pair.
- Continuous RNG (CRNG) Test each time the module produces random data, as per the FIPS 140-2 standard. The CRNG test is performed on all approved and non-approved PRNGs (HMAC DRBG, HASH DRBG, CTR DRBG, FIPS186 PRNG, non-approved entropy source).

1.6.3 Mitigation of Other Attacks

RSA, EC, DSA and KAS-FFC-SSC key operations implement blinding by default. Blinding is a reversible way of modifying the input data, so as to make the operations immune to timing attacks. Blinding has no effect on the algorithm other than to mitigate attacks on the algorithm.

RSA, EC, DSA and KAS-FFC-SSC 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.

For other types of timing attacks the module implements time invariant comparisons and operations, for example, PKCS #1 unpadding, HMAC verify, and RSA verify.

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

2 Secure Operation of the Module

The following guidance must be followed in order to operate the module in a FIPS 140-2 mode of operation, in conformance with FIPS 140-2 requirements.

> **Note:** The module operates as a Validated Cryptographic Module only when the rules for secure operation are followed.

2.1 Module Configuration

To operate the module in compliance with FIPS 140-2 Level 1 requirements, the module must be loaded using the following method:

`com.rsa.crypto.jcm.ModuleLoader.load()`

The `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 140-2 Level 1 compliant way.

Once the load method has been successfully called, the module is operational.

2.2 Security Roles, Services and Authentication Operation

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

The mode argument must be the value `FIPS140Context.MODE_FIPS140`.

To retrieve the current JCM FIPS 140-2 mode, call `FIPS140Context.getMode()`.

The available role values are the constants `FIPS140Context.ROLE_CRYPTO_OFFICER` and `FIPS140Context.ROLE_USER`.

No role authentication is required for operation of the module in Security Level 1 mode. When in Security Level 1 mode, invocation of methods which are particular to Security Level 2 Roles, Services and Authentication will result in an error.

2.3 Crypto User Guidance

This section provides guidance to the module user to ensure that the module is used in a FIPS 140-2 compliant way.

Section 2.3.1 provides algorithm-specific guidance. The requirements listed in this section are not enforced by the module and must be ensured by the module user.

Section 2.3.2 provides guidance on obtaining assurances for Digital Signature Applications.

Section 2.3.3 provides guidance on obtaining assurances for Key Transport Applications.

Section 2.3.4 provides guidance on the entropy requirements for secure key generation.

Section 2.3.5 provides general crypto user guidance.

2.3.1 Crypto User Guidance on Algorithms

The following table lists the algorithm requirements for FIPS 140-2 mode of operation:

Table 9 Algorithm Requirements for FIPS 140-2 Mode of Operation

Algorithm	Guidance
DRBG	
	• When an approved algorithm requires a DRGB to perform an operation, an approved DRBG algorithm must be used. For example, when initializing an approved signature algorithm, an approved DRBG such as HMAC DRBG must be used. • When using an approved DRBG, the number of bytes of seed key input must be equivalent to or greater than the security strength of the keys the caller wishes to generate. For example, a 256-bit or higher seed key input when generating 56-bit AES keys. • Since the module does not modify the output of an Approved DRBG, any generated symmetric keys or seed values are created directly from the output of the Approved DRBG.

Table 9 Algorithm Requirements for FIPS 140-2 Mode of Operation (continued)

Algorithm	Guidance
GCM Mode Ciphers	
	• When using GCM feedback mode for symmetric encryption, the authentication tag length and authenticated data length may be specified as input parameters, but the IV must not be specified. It must be generated internally. • Where the module is powered down, a new key must be used for AES GCM encryption/decryption. • GCM with a partial IV supplied to the module is approved only when used within a TLS 1.2 protocol implementation. Note that the module conforms to IG A.5 Scenario 1 for TLS, sub-scenario ii. • The AES-GCM cipher, when used for symmetric encryption purposes other than TLS, must use an IV in one of the two possible ways, to comply with SP 800-38D: – Allow the module to generate the IV 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 where: – 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. – Generate at least 12 bytes of IV using an Approved DRBG, and input the IV to the cipher at initialization time using the `RAW_IV` parameter. • The AES-GCM cipher used for the TLS protocol as the cipher implementation complies with SP 800-52 and is compatible with RFC 5288 with the following conditions: – The IV is configured as follows: – 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 FIPS 140-2 IG A.5 and is positioned in the name field of the IV as required in FIPS 140-2 IG A.5 Scenario 3. – The remaining eight bytes of IV, referred to as `nonce_explicit` in RFC 5288, are generated deterministically by the module using the 64-bit counter used for the invocation field described above. – 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. – The TLS session is aborted if the keys for the client and server negotiated in the handshake process, `client_write_key` and `server_write_key`, are identical.

Table 9 Algorithm Requirements for FIPS 140-2 Mode of Operation (continued)

Algorithm	Guidance
HMACs	
	• The key length for an HMAC generation or verification must be between 112 and 4096 bits, inclusive. • For HMAC verification, a key length greater than or equal to 80 and less than 112 is allowed for legacy-use.
HMAC-Based Extract-and-Expand Key Derivation Function	
	• A FIPS 140-2 approved HMAC must be used for extract and expand operations. • 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.
One-Step Key Derivation Function	
	• A FIPS 140-2 approved hash function must be used to derive key materials. • When selecting an 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.
TLS PRF Key Derivation Function	
	• TLS 1.2 PRF KDF is allowed only when the following conditions are satisfied: – The KDF is performed in the context of the TLS protocol. – HMAC is as specified in FIPS 198-1. – P_HASH uses either SHA-256, SHA-384, or SHA-512. For more information, see SP 800-135 Rev. 1. The TLS protocols have not been tested by the CAVP and CMVP.
Parameter Generation	
	• When using an Approved DRBG to generate KAS-FFC-SSC or DSA parameters, the requested DRBG must have a security strength at least as great as the security strength of the parameters being generated. That means that an Approved DRBG with an appropriate strength must be used. For more information on requesting the DRBG security strength, see the relevant API *Javadoc*.

Table 9 Algorithm Requirements for FIPS 140-2 Mode of Operation (continued)

Algorithm	Guidance
Key Agreement	

	The following requirements ensure that the module does not perform any key agreement functionality that does not comply with SP 800-56A Rev. 3 when running in FIPS 140-2 mode:
	Obtain domain parameters and assurance of the domain parameter validity:
	• 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 800-56A Rev. 3 Appendix D.
	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 a FIPS-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 with 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.

Table 9 Algorithm Requirements for FIPS 140-2 Mode of Operation (continued)

Algorithm	Guidance
Key Agreement (continued)	
	• For schemes that generate an FFC key pair from selected parameters, the key pair must not be used to generate a digital signature. 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, must be used. – 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: – Approved key confirmation technique(s) as specified in SP 800-56A Rev. 3 must be used. – 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 before it is used for any other purpose. – The module provides the KeyConfirmation API class for the key confirmation requirements in SP 800-56A Rev. 3.

Table 9 Algorithm Requirements for FIPS 140-2 Mode of Operation (continued)

Algorithm	Guidance
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.
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 2048 bits, however 1024 bits is allowed for legacy-use only. RSA keys shall have a public exponent of an odd number, equal to or greater than 65537. • The SHA1 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).

Table 9 Algorithm Requirements for FIPS 140-2 Mode of Operation (continued)

Algorithm	Guidance
Password-based Key Derivation	
	• Keys generated using PBKDF2 shall only be used in data storage applications. • Minimum Password Length: 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: $L = \lceil \log_2 S / \log_2 N \rceil$ The following provides examples for a password used by PBKDF2 where $S = 4.32 \times 10^{20}$: **Character Set** — **N** — **L** Case sensitive (a-z, A-Z) — 52 — 13 Case sensitive alpha numeric — 62 — 12 All ASCII printable characters except space — 94 — 11 • A password of the strength S can be guessed at random with the probability of 1 in 2^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. • Derived keys can be used as specified in NIST SP 800-132, Section 5.4, options 1 and 2.
XTS Mode Ciphers	
	• AES in XTS mode is approved only for hardware storage applications. • The two keys used for XTS must be checked to ensure they are different. This check is performed automatically by the module.

2.3.2 Crypto User Guidance on Obtaining Assurances for Digital Signature Applications

The module provides support for the FIPS 186-4 standard for digital signatures. The following gives an overview of the assurances required by FIPS 186-4. NIST Special Publication 800-89: "Recommendation for Obtaining Assurances for Digital Signature Applications" provides the methods to obtain these assurances.

The following tables describe the FIPS 186-4 requirements for signatories and verifiers and the corresponding module capabilities and recommendations.

Table 10 Signatory Requirements

FIPS 186-4 Requirement	Module Capabilities and Recommendations
Obtain appropriate DSA and ECDSA parameters when using DSA or ECDSA.	The generation of DSA parameters is in accordance with the FIPS 186-4 standard for the generation of probable primes. For ECDSA, use the NIST recommended curves as defined in section 2.3.1 Crypto User Guidance on Algorithms.
Obtain assurance of the validity of those parameters.	The module provides APIs to validate DSA parameters for probable primes as described in FIPS 186-4. For ECDSA, use the NIST recommended curves as defined in section 2.3.1 Crypto User Guidance on Algorithms. For the JCM API, `AlgParamGenerator.verify()`
Obtain a digital signature key pair that is generated as specified for the appropriate digital signature algorithm.	The module generates the digital signature key pair according to the required standards. Choose a FIPS-Approved DRBG like HMAC DRBG to generate the key pair.
Obtain assurance of the validity of the public key.	The module provides APIs to explicitly validate the public key according to SP 800-89. For the JCM API, `PublicKey.isValid(SecureRandom secureRandom)`
Obtain assurance that the signatory actually possesses the associated private key.	The module verifies the signature created using the private key, but all other assurances are outside the scope of the module.

Table 11 Verifier Requirements

FIPS 186-4 Requirement	Module Capabilities and Recommendations
Obtain assurance of the signatory's claimed identity.	The module verifies the signature created using the private key, but all other assurances are outside the scope of the module.
Obtain assurance of the validity of the domain parameters for DSA and ECDSA.	The module provides APIs to validate DSA parameters for probable primes as described in FIPS 186-4. For the JCM API, `AlgParamGenerator.verify()` For ECDSA, use the NIST recommended curves as defined in section 2.3.1 Crypto User Guidance on Algorithms.

Table 11 Verifier Requirements (continued)

FIPS 186-4 Requirement	Module Capabilities and Recommendations
Obtain assurance of the validity of the public key.	The module provides APIs to explicitly validate the public key according to SP 800-89. For the JCM API, `PublicKey.isValid(SecureRandom secureRandom)`
Obtain assurance that the claimed signatory actually possessed the private key that was used to generate the digital signature at the time that the signature was generated.	Outside the scope of the module.

2.3.3 Crypto User Guidance on Obtaining Assurances for Key Transport Applications

The module provides support for the NIST SP800.56B recommendations for key transport. NIST Special Publication 800-56B Revision 1: “Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography” describes the requirements for obtaining these assurances.

The following table describes the SP 800-56B recommendations for key transport and the corresponding module capabilities and recommendations, as the module only supports the primitives and underlying functions of SP 800-56B, not the full scheme.

Table 12 Key Transport Recommendations

NIST SP 800-56B Recommendations	Module Capabilities and Recommendations
Assurance of Key-Pair Validity	The module provides APIs to explicitly validate an RSA Key Pair according to SP 800.56B. The JCM API available is: `KeyPair.validate(AlgInputParams, SecureRandom)`. The parameters object can be supplied with `SECURITY_STRENGTH` and `KEY_BITS` inputs. This API performs both a pairwise consistency test and a key pair validation according to `“rsakpv1-crt”` and `“crt_pkv”` methods.
Assurance of Public Key Validity	The module provides APIs to explicitly validate the RSA public key according to SP 800.56B and SP 800-89. The JCM API available is: `KeyPair.validate(AlgorithmParams, SecureRandom)`
Assurance of Possession of Private Key	The module supports Key Confirmation for providing assurance of possession of a private key in a key transport scheme. The JCM API available is: `KeyConfirmation`.

2.3.4 Crypto User Guidance on Key Generation and Entropy

No assurance is given for the minimum strength of generated keys. The JCM provides the HMAC DRBG, CTR DRBG and Hash DRBG implementations for key generation.

When generating secure keys, 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 key 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 the key generation operation.

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.

Table 13 Generated Key Sizes and Strength

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	112, 128	112, 128
DSA Key Pair	2048, 3072	112, 128	112, 128
EC Key Pair	224, 256, 384, 521	112, 128, 192, 256	112, 128, 192, 256

2.3.5 General Crypto User Guidance

JCM users should take care to zeroize CSPs when they are no longer needed. For more information on clearing sensitive data, see section 1.4.4 Key Zeroization and the relevant API *Javadoc*.

2.4 Crypto Officer Guidance

The Crypto Officer is responsible for installing the module. Installation instructions are provided in the *Dell BSAFE Crypto-J Installation Guide*.

The Crypto Officer is responsible for loading the module, as specified in section 2.1 Module Configuration.

2.5 Operating the Cryptographic Module

Both FIPS and non-FIPS algorithms are available to the operator. In order to operate the module in the FIPS-Approved mode, all rules and guidance provided in Secure Operation of the Module **must** be followed by the module operator. The module **does not** enforce the FIPS 140-2 mode of operation.

3 Acronyms

The following table lists the acronyms used with the JCM and their definitions.

Table 14 Acronyms used with the JCM

Acronym	Definition
3DES	Refer to Triple-DES
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.
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.

Table 14 Acronyms used with the JCM (continued)

Acronym	Definition
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.
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.
ECDHC	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

Table 14 Acronyms used with the JCM (continued)

Acronym	Definition
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.
HMAC	Keyed-Hashing for Message Authentication Code.
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.
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	Password-Based Key Derivation Function.

Table 14 Acronyms used with the JCM (continued)

Acronym	Definition
PBKDF2	A method of password-based key derivation, originally defined in RFC 2898, which applies a MAC algorithm to derive the key. In RFC 2898 the PRF used by PBKDF2 is specified as SHA-1. SP 800-132 approves PBKDF2 where the PRF may be any FIPS approved hash function. In this document PBKDF2 represents the expanded specification provided in SP 800-132.
PC	Personal Computer.
Poly1305	A cryptographic MAC standardized in RFC 7539.
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 (SHA-256, SHA-384 and SHA-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.

Table 14 Acronyms used with the JCM (continued)

Acronym	Definition
Shamir's 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 `k` of the `n` parts are sufficient to reconstruct the original secret.
TDES	Refer to Triple-DES.
TLS	Transport Layer Security.
Triple-DES	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.
XTS	XEX-based Tweaked Codebook mode with ciphertext stealing. A mode of encryption used with AES.

4 Change Summary

May 2023	• Updated section 1.5.2 header. • Changed some Diffie-Hellman references to a corresponding KAS reference. • Specified the cryptographic standard for some algorithms. • Updated some TLS references. • Added reference to section 1.4.2 Key Assurance in section 1.4.6 Key Access.

© 2024 Dell Inc. or its subsidiaries. All rights reserved. Dell, BSAFE and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be trademarks of their respective owners.